

SSSSSSSSSSSS	DDDDDDDDDDDD	AAAAA
SSSSSSSSSSSS	DDDDDDDDDDDD	AAAAA
SSSSSSSSSSSS	DDDDDDDDDDDD	AAAAA
SSS	DDD	AAA
SSS	DDD	AAA
SSS	DDD	AAA
SSS	DDD	AAA
SSS	DDD	AAA
SSS	DDD	AAA
SSSSSSSSSS	DDD	AAA
SSSSSSSSSS	DDD	AAA
SSSSSSSSSS	DDD	AAA
SSS	DDD	AAAAA
SSS	DDD	AAAAA
SSS	DDD	AAAAA
SSS	DDD	AAA
SSS	DDD	AAA
SSS	DDD	AAA
SSSSSSSSSSSS	DDDDDDDDDDDD	AAA
SSSSSSSSSSSS	DDDDDDDDDDDD	AAA
SSSSSSSSSSSS	DDDDDDDDDDDD	AAA

```

CCCCCCCCC LL UU UU SSSSSSSS TTTTTTTTT EEEEEEEEE RRRRRRR
CCCCCCCCC LL UU UU SSSSSSSS TTTTTTTTT EEEEEEEEE RRRRRRR
CC CC LL LL SS SS TT EE RR RR RR
CC CC LL LL SS SS TT EE RR RR RR
CC CC LL LL SS SS TT EE RR RR RR
CC CC LL LL SS SS TT EE RR RR RR
CC CC LL LL SS SS TT EE RR RR RR
CC CC LL LL SS SS TT EE RR RR RR
CC CC LL LL SS SS TT EE RR RR RR
CCCCCCCCC LLLLLLLLLL UUUUUUUUU SSSSSSSS TTTTTTTTT EEEEEEEEE RRRRRRR
CCCCCCCCC LLLLLLLLLL UUUUUUUUU SSSSSSSS TTTTTTTTT EEEEEEEEE RRRRRRR
LLLLLLLLL IIIIII SSSSSSSS
IIIIII IIIIII SSSSSSSS
II II SS
II II SS
II II SS
II II SS
II II SSSSSS
II II SSSSSS
II II SS
II II SS
II II SS
II II SS
II II SS
IIIIII SSSSSSSS
IIIIII SSSSSSSS

```


(1)	2	COPYRIGHT NOTICE
(1)	29	PROGRAM DESCRIPTION
(2)	56	declarations
(3)	77	storage definitions
(4)	130	read-only data definitions
(5)	182	show_cluster --- display structures relevant to vaxclusters
(6)	280	cluster_summary --- summary sheet for the club and csbs
(7)	379	display_club --- display cluster block (CLUB)
(8)	419	cluster_block_data block tables & action routines
(9)	589	display_clufcb --- display cluster failover control block (CLUFCB)
(10)	630	display_cludcb --- display cluster quorum disk control block
(11)	680	cluster_failover_control block tables & action routines
(12)	697	cluster_quorum_disk_control block tables & action routines
(13)	717	display_csb --- display cluster system block (CSB)
(14)	781	cluster_system_block tables & action routines
(15)	875	show_scs --- display system communications (SCS) data structures
(16)	912	scs_summary --- display system communications (SCS) summary
(17)	1029	display_sb_pbs --- display all system and path blocks
(18)	1063	show_connections --- display all connection descriptor tables (CDT)
(19)	1201	display_sumline --- display a line of the cdt summary page
(20)	1268	state_translate --- translate cdt state values to names
(21)	1305	find_procname --- Find the local process name.
(22)	1365	remote_node --- find the remote node name
(23)	1400	display_cdt --- display a connection descriptor table
(24)	1523	cdt_byaddr --- display the cdt requested by the user
(25)	1583	connection_descriptor tables & action routines
(26)	1668	show_rspid --- display RDT entries
(27)	1794	display_rd_entry --- display an entry in the response descriptor table
(28)	1858	show_ports --- display all port descriptor tables (PDT)
(29)	1965	display_pdt --- display a port descriptor table
(30)	2031	pdt_byaddr --- display the pdt requested by the user
(31)	2089	port_descriptor tables & action routines

CLUSTER
V04-000

SHOW CLUSTER INFORMATION

C 15

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 1
(1)

```
0000 1 .TITLE CLUSTER SHOW CLUSTER INFORMATION
0000 2 .SBTTL COPYRIGHT NOTICE
0000 3 .IDENT 'V04-000'
0000 4 :
0000 5 :*****
0000 6 :*
0000 7 :* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 :* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 :* ALL RIGHTS RESERVED.
0000 10 :*
0000 11 :* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 :* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 :* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 :* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 :* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 :* TRANSFERRED.
0000 17 :*
0000 18 :* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 :* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 :* CORPORATION.
0000 21 :*
0000 22 :* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 :* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 :*
0000 25 :*
0000 26 :*****
0000 27 :
```



```
0000 29      .SBTTL PROGRAM DESCRIPTION
0000 30      :++
0000 31      FACILITY
0000 32      :
0000 33      SYSTEM DUMP ANALYZER
0000 34      :
0000 35      ABSTRACT
0000 36      :
0000 37      THIS MODULE CONTAINS THE ROUTINES NECESSARY TO DISPLAY THE
0000 38      VAXCLUSTER DATA STRUCTURES. IT PROVIDES SUPPORT FOR THE COMMANDS,
0000 39      SHOW CLUSTER, SHOW RSPID, SHOW CONNECTIONS, AND SHOW PORTS.
0000 40      :
0000 41      ENVIRONMENT
0000 42      :
0000 43      NATIVE MODE, USER MODE
0000 44      :
0000 45      AUTHOR
0000 46      :
0000 47      ELLEN M. BATBOUTA, MAY 1984
0000 48      :
0000 49      V03-001 EMD0110      Ellen M. Batbouta      16-JUL-1984
0000 50      Allocate storage dynamically for the cdl and the rdt since
0000 51      their sizes depend on sysgen parameters. Display closed
0000 52      cdt's only if the /address qualifier is specified on the
0000 53      command. Also fix a few minor problems with the displays.
0000 54      :
```



```
0000 56 .sbtll declarations
0000 57 :
0000 58 : symbol definitions
0000 59 :
0000 60 $cdldef ; Connection Descriptor List (CDL)
0000 61 $cdtdef ; SCS Connection Descriptor Table (CDT)
0000 62 $cdrpdef ; Class Driver Request Packet (CDRP)
0000 63 $clubdef ; Cluster Block (CLUB)
0000 64 $cludcbdef ; Cluster Quorum Disk Control Block (CLUDCB)
0000 65 $csbdef ; Cluster System Block (CSB)
0000 66 $ddbdef ; Device Data Block (DDB)
0000 67 $dyndef ; Dynamic Storage Type Definitions
0000 68 $pbdef ; Path Block (PB)
0000 69 $pdtdef ; Port Descriptor Table (PDT)
0000 70 $rddef ; SCS Response Descriptor Format
0000 71 $rtddef ; SCS Response Descriptor Table
0000 72 $sbdef ; System Block (SB)
0000 73 $sdirdef ; SCS Directory Entry (SDIR)
0000 74 $tpadef ; TPARSE definitions
0000 75 $ucbdef ; Unit Control Block (UCB)
```



```
0000 77 .sbtll storage definitions
0000 78 :
0000 79 : storage definitions
0000 80 :
0000 81 :
00000000 82 .psect sdadata,noexe,wrt
0000 83
00000004 0000 84 cdl: .blk1 1 ; to contain address of local cdl
0004 85 ; Connections Descriptor List (CDL)
0004 86 cdl_size:
00000008 0004 87 .blk1 1 ; to contain size of cdl
0008 88
000000A8 0008 89 cdt: .blkb cdt$c_length ; connection descriptor table (CDT)
00A8 90
00000250 00A8 91 club: .blkb club$c_length ; Cluster Block (CLUB)
0250 92
000002FC 0250 93 csb: .blkb csb$c_length ; Cluster System Block (CSB)
02FC 94
00000525 02FC 95 cludcb: .blkb cludcb$c_length ; Cluster Quorum Disk Control Block
0525 96
00000609 0525 97 pdt: .blkb pdt$c_length ; Port Descriptor Table (PDT)
0609 98
00000639 0609 99 directory:
0609 100 .blkb sdir$c_length ; SCS directory entry
0639 101
0000063D 0639 102 rdt: .blk1 1 ; to contain address of local rdt
063D 103
063D 104 rdt_size:
00000641 063D 105 .blk1 1 ; to contain size of rdt
0641 106 wait_cdrp:
00000000 0641 107 .long 0 ; cdrp in rdt wait queue
0645 108
0645 109 sblock:
000006A5 0645 110 .blkb sb$c_length ; System Block (SB)
06A5 111
000006B5 06A5 112 node: .blkb sb$s_nodename ; node name in system block (SB)
06B5 113
000006C5 06B5 114 procname: .blkb 16 ; to hold local/remote process name
06C5 115
000006D9 06C5 116 driver_name: .blkb 20 ; driver name
06D9 117
000006ED 06D9 118 device_name: .blkb 20 ; device name
06ED 119
06ED 120 tim_buffer:
000006F5 06ED 121 .blk1 2 ; buffer to hold date/time stamp
06F5 122 csid:: .long 0 ; cluster system id
00000000 06F5 123
06F9 124
06F9 125 cdt_spcfy::
00000000 06F9 126 .long 0 ; flag to specify if /connection
06FD 127 ; qualifier was present in command
06FD 128
```



```
06FD 130 .sbtll read-only data definitions
06FD 131 :
06FD 132 : read-only data definitions
06FD 133 :
06FD 134 :
00000000 135 .psect cluster,exe,nowrt,long
0000 136 .default displacement,long
0000 137
0000 138 club_summary:
0000 139 table club$V_,<qf_dynvote,qf_vote,quorum,transition>
0028 140
0028 141 csb_summary:
0028 142 table csb$V_,<long_break,member,removed,qf_same,qf_active>
0058 143
0058 144 csb_states:
0058 145 table csb$K_,<open,status,reconnect,new,connect,accept,disconnect,-
0058 146 reaccept,wait,dead,local>
00B8 147
00B8 148 csb_status:
00B8 149 table csb$V_,<long_break,member,removed,qf_same,cluster,qf_active,-
00B8 150 shutdown,locked,selected,local,status_rcvd,send_status>
0120 151
0120 152 fcb_status:
0120 153 table clufcb$V_,<active,pending,sync_node,fkb_busy>
0148 154
0148 155 club_flags:
0148 156 table club$V_,<cluster,qf_active,shutdown,sts_pphase,sts_ph0,-
0148 157 sts_ph1b,sts_ph1,sts_ph2,fkb_busy,unlock,no_form,-
0148 158 init,backout,lost_cnx,qf_failed_node,qf_vote,-
0148 159 qf_newvote,adj_quorum,quorum,transition,qf_dynvote>
01F8 160
01F8 161 cludcb_state:
01F8 162 table cludcb$V_,<qs_not_ready,qs_ready,qs_active,qs_cluster,qs_vote>
0228 163
0228 164 cludcb_flags:
0228 165 table cludcb$V_,<qf_tim,qf_rip,qf_wip,qf_error,qf_cspack>
0258 166
0258 167 cdt_state:
0258 168 table cdt$C_,<closed,listen,open,disc_ack,disc_rec,disc_sent,-
0258 169 disc_mtch,con_sent,con_ack,con_rec,accp_sent,rej_sent,-
0258 170 vc_fail>
02C8 171
02C8 172 cdt_blkstate:
02C8 173 table cdt$C_,<con_pend,accp_pend,rej_pend,disc_pend,cr_pend,dcr_pend>
0300 174
0300 175 pdt_type:
0300 176 table pdt$C_,<pa,pu,pe,ps>
0328 177
0328 178 port_char:
0328 179 table pdt$V_,<snghost>
0338 180
```



```
0338 182 .sbttl show_cluster --- display structures relevant to vaxclusters
0338 183 :---
0338 184 :
0338 185 show_cluster
0338 186 :
0338 187 This is the main routine whose purpose is to provide information
0338 188 on vaxclusters. Several structures are displayed. The order
0338 189 is as follows:
0338 190 list of cluster system blocks (CSBs)
0338 191 the cluster block (CLUB)
0338 192 the cluster failover control block (CLUFCB)
0338 193 the cluster quorum disk control block (CLUDCB)
0338 194 display a csb for each node in the cluster
0338 195
0338 196 Inputs:
0338 197
0338 198 AP = pointer to TPARSE block
0338 199 CSID = cluster system id (CSID)
0338 200
0338 201 Outputs:
0338 202
0338 203 Vaxcluster data structures ( as listed above) are shown
0338 204 :---
0338 205 .enable lsb
0338 206 show_cluster::
OFFC 0338 207 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
033A 208
033A 209 subhd <VAXcluster data structures> ; set heading
0347 210
0347 211 getmem @clu$gl_club,r5 ; get address of club
0357 212 blbs r0,5$ ; branch if able to read it
035A 213 brw 20$ ; branch because of error
52 000000A8'EF 9E 035D 214 5$:
035D 215 movab club,r2 ; will contain local copy of club
0364 216 getmem (r1),(r2),#club$c_length ; move club to local storage
69 50 E9 0375 217 blbc r0,20$ ; check for error
000006F5'EF B5 0378 218 tstw csid ; check to see if csid in command
65 12 037E 219 bneq locate_csb ; display csb of this csid and exit
0380 220
0380 221 pushl r5 ; address of club
52 DD 0382 222 pushl r2 ; pass address of local club
0000043F'EF 01 FB 0384 223 calls #1,cluster_summary ; display list of csb's
038B 224
038B 225 pushl r5 ; address of club
52 DD 038D 226 pushl r2 ; pass address of local club
000005A8'EF 01 FB 038F 227 calls #1,display_club ; display cluster block
0396 228
010C C2 9F 0396 229 pushab club$b_clufcb(r2) ; address of fcb in local storage
010C C5 9F 039A 230 pushab club$b_clufcb(r5) ; failover control block
0000093D'EF 01 FB 039E 231 calls #1,display_clufcb ; display it
03A5 232
00B4 C2 D5 03A5 233 tstl club$l_cludcb(r2) ; cludcb exists?
08 13 03A9 234 beql 6$ ; equal, does not exist
00B4 C2 DD 03AB 235 pushl club$l_cludcb(r2) ; quorum disk control block
000009D6'EF 01 FB 03AF 236 calls #1,display_cludcb ; display it
03B6 237 6$:
54 65 DE 03B6 238 moval club$l_csbqfl(r5),r4 ; address of csb queue
```



```

        62 54 D1 03B9 239      cmpl    r4,club$l_csbqfl(r2)      ; check if queue empty
        53 23 13 03BC 240      beql    20$                    ; equal, then empty, so exit
        53 62 D0 03BE 241      movl    club$l_csbqfl(r2),r3      ; queue not empty
00000B72'EF 53 53 DD 03C1 242 10$:  pushl    r3                ; pass it to routine
        53 01 FB 03C3 243      calls   #1,display_csb          ; display this csb
        53 00 C0 03CA 244      addl2   #csb$l_sysqfl,r3         ; check for another csb
        05 50 E9 03CD 245      getmem   (r3),r3                ; read field in queue
        53 54 D1 03D9 246      blbc    r0,20$                  ; are we able to read it?
        53 E0 12 03DC 247      cmpl    r4,r3                    ; check to see if at end of queue
        50 01 D0 03DF 248      bneq    10$                      ; not equal, another csb exists
        04 03E1 249 20$:      movl    #1,r0                    ; finished! - return success
        04 03E4 250      ret
        03E5 251
        03E5 252
        03E5 253
        03E5 254 locate_csb:
        54 65 DE 03E5 255      movl    club$l_csbqfl(r5),r4      ; start of queue
        62 54 D1 03E8 256      cmpl    r4,club$l_csbqfl(r2)    ; is queue empty
        53 F4 13 03EB 257      beql    20$                    ; equal, yes so exit
        53 62 D0 03ED 258      movl    club$l_csbqfl(r2),r3      ; first entry in queue
56 53 0000004C 8F C1 03F0 259 40$:  addl3   #csb$l_csid,r3,r6      ; point to csid in csb
        DD 50 E9 03F8 260      getmem   (r6)                    ; read in csid value
000006F5'EF 53 51 B1 0401 261      blbc    r0,20$                  ; exit if can not read
        53 19 13 0404 262      cmpw    r1,csid                  ; right csb?
        53 00 C0 040B 263      beql    50$                    ; equal, yes so display
        C2 50 E9 040D 264      addl2   #csb$l_sysqfl,r3         ; point to next entry in queue
        53 54 D1 0410 265      getmem   (r3),r3                ; read it in
        BD 13 041C 266      blbc    r0,20$                  ; exit if not possible
        CA 11 041F 267      cmpl    r4,r3                    ; end of queue yet
        0422 268      beql    20$                      ; equal, yes so exit
        0424 269      brb    40$                      ; get next csb
        0426 270 50$:      skip    page
        53 DD 0426 271      pushl    r3                        ; next screen
00000B72'EF 53 01 FB 042D 272      calls   #1,display_csb          ; actual address of csb
        000006F5'EF 53 D4 042F 273      clrl    csid              ; display
        FFA2 31 0436 274      brw    20$                      ; reinitialize
        043C 275      .dsabl    lsb                          ; and exit with success
        043F 276
        043F 277
        043F 278
```



```
043F 280 .sbtcl cluster_summary --- summary sheet for the club and csbs
043F 281 ---
043F 282
043F 283 cluster_summary
043F 284
043F 285 This routine outputs a brief summary of the cluster block (CLUB)
043F 286 and of each cluster system block (CSB). There exists
043F 287 one csb per node in the cluster and one club for the cluster.
043F 288
043F 289 Inputs:
043F 290
043F 291 4(ap) = address of club in local storage
043F 292 8(ap) = actual address of club
043F 293
043F 294 Outputs:
043F 295
043F 296
043F 297 ---
043F 298 cluster_summary:
OFFC 043F 299 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
54 04 AC DO 0441 300
0445 301 movl 4(ap),r4 ; club in local storage
0445 302
0445 303 First display a few important fields in the cluster block (CLUB)
0445 304
0445 305 skip page
044C 306 print 0,<!-- --- VAXcluster Summary --->
0459 307 skip 1
0462 308 print 0,<!-- Quorum Votes Quorum Disk Votes Status Summary>
046F 309 print 0,<!-- ----->
047C 310
047C 311 alloc 80,r2 ; allocate output buffer
7E 1C A4 DO 048E 312 movl club$l_flags(r4),-(sp) ; bit mask to translate
00000000'EF FB 9F 0492 313 pushab club_summary ; address of definition table
049D 314 calls #2,translate_bits ; translate bits to names
049D 315
049D 316 pushl sp ; address of string descriptor
7E 00AE C4 DD 049F 317 movzwl club$w_qdvotes(r4),-(sp) ; quorum disk votes
7E 22 A4 3C 04A4 318 movzwl club$w_votes(r4),-(sp) ; cluster votes
7E 20 A4 3C 04A8 319 movzwl club$w_quorum(r4),-(sp) ; cluster quorum
04AC 320 print 4,<!-- !4<!-- !4<!-- !4<!-- !AS>
5E 00000058 8F C0 04B9 321 addl2 #88,sp ; clean up stack
04C0 322
04C0 323 Now to actually display a list of csb's and a little information about
04C0 324 each one. (A little knowledge never hurt anyone, right?)
04C0 325
04C0 326 skip 1
04C9 327 print 0,<!-- !_!_--- CSB list --->
04D6 328 skip 1
04DF 329 print 0,<Address Node CSID Votes State Status>
04EC 330 print 0,<----->
04F9 331 skip 1
0502 332
0502 333 Header information complete - now time to loop through the queue of csb's
0502 334 in the cluster block (club)
0502 335
0502 336 assume club$l_csbqfl eq 0
```



```

      64 08 AC D1 0502 337      cmpl      8(ap),club$l_csbqfl(r4)      ; check for empty queue
      03 12 0506 338      bneq      20$      ; not equal, entry in queue
      0099 31 0508 339      brw      done      ; otherwise, this display is done
      56 64 D0 050B 340 20$:      movl      club$l_csbqfl(r4),r6      ; get address of csb
      DE 050E 341      Loop:      moval      csb,r7      ; local storage for csb
57 00000250'EF 0515 344      getmem      (r6),(r7),#csb$c_length      ; read entire csb
      7B 50 E9 0526 345      blbc      r0,done      ; if not able to read, exit
      0529 346
      0529 347      alloc      80      ; alloc buffer for translation
      7E 60 A7 D0 0538 348      movl      csb$l_status(r7),-(sp)      ; bit mask to translate
      FAE8 CF 9F 053C 349      pushab      csb_summary      ; bit definition table
00000000'EF 02 FB 0540 350      calls      #2,translate_bits      ; translate bits to names
      5E DD 0547 351      pushl      sp      ; names for status bits
      0549 352
      52 43 A7 9A 0549 353      movzbl      csb$b_state(r7),r2      ; bit mask to translate
53 FB07 CF 9E 054D 354      movab      csb_states,r3      ; state translation table
00000000'GF 16 0552 355      jsb      g^translate_address      ; translate value to names
      02 13 0558 356      beql      10$      ; branch if translation failed
      50 DD 055A 357      pushl      r0      ; names for states
      055C 358 10$:
      7E 50 A7 3C 055C 359      movzwl      csb$w_votes(r7),-(sp)      ; votes held by node
      4C A7 DD 0560 360      pushl      csb$l_csid(r7)      ; Cluster System Id
      0563 361
      52 000006A5'EF 9E 0563 362      movab      node,r2
53 68 A7 00000044 8F C1 056A 363      addl3      #sb$t_nodename,csb$l_sb(r7),r3      ; point to nodename
      0573 364      getmem      (r3),(r2),#sb$s_nodename      ; read in nodename
      52 DD 0580 365      pushl      r2      ; node
      56 DD 0582 366      pushl      r6      ; address of csb
      0584 367      print      6,<!XL !6<!AC!> !XL !3<!UW!>!10< !AC!> !AS>
5E 00000058 8F C0 0591 368      addl2      #88,sp      ; clean up stack
      0598 369
      08 AC 67 D1 0598 370      cmpl      csb$l_sysqfl(r7),8(ap)      ; check for end of csbs
      06 13 059C 371      beql      done      ; equal, at end
      56 67 D0 059E 372      movl      csb$l_sysqfl(r7),r6      ; address of next csb
      FF6A 31 05A1 373      brw      loop      ; loop to display
      05A4 374
      05A4 375      done:
      50 01 D0 05A4 376      movl      #1,r0
      04 05A7 377      ret
```



```
05A8 379 .sbtcl display_club --- display cluster block (CLUB)
05A8 380 ---
05A8 381
05A8 382 display_club
05A8 383
05A8 384 This routine displays the cluster block. There exists
05A8 385 one club per cluster.
05A8 386
05A8 387 Inputs:
05A8 388
05A8 389 4(ap) = address of club in local storage
05A8 390 8(ap) = actual address of club
05A8 391
05A8 392 Outputs:
05A8 393 The CLUB is displayed.
05A8 394
05A8 395 ---
05A8 396 display_club:
OFFC 05A8 397 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
05AA 398
05AA 399 ensure 20 ; make sure at least 20 lines on screen
54 04 AC DO 05C2 400 movl 4(ap),r4 ; club in local storage
08 AC DD 05C6 401 pushl 8(ap) ; actual address of club
05C9 402 print 1,<!_!_ --- Cluster Block (CLUB) !XL --->
05D6 403 skip 1
05DF 404 alloc 80,r5 ; 80 byte output buffer
1C A4 DD 05F1 405 pushl club$l_flags(r4) ; flags in club
FB50 CF 9F 05F4 406 pushab club_flags ; bit definition table
00000000'EF 02 FB 05F8 407 calls #2,translate_bits ; translate bits to names
55 DD 05FF 408 pushl r5 ; names defining flags
1C A4 DD 0601 409 pushl club$l_flags(r4) ; flags in club
0604 410 print 2,<Flags: !XL !AS> ; display translated flags
5E 00000058 8F C0 0611 411 skip 1
061A 412 addl2 #88,sp ; clean up stack
0621 413 print_columns -
0621 414 (r4),8(ap),-
0621 415 club_col_1,club_col_2 ; display the club
0639 416 status success
04 0640 417 ret
```



```
0641 419 .sbttl cluster block data block tables & action routines
0641 420 :
0641 421 : PRINT_COLUMNS table for CLUB displays
0641 422 :
0641 423 :
0641 424 club_fao_6bytes:
0641 425 string <!* !XW!XL>
0653 426
0653 427 club_2words:
0653 428 string <!10UW!UW>
0664 429
0664 430 club_col_1:
0664 431 column_list
0664 432 club$, 21, 12, 4, <-
0664 433 <<Quorum/Votes>, quor_vote, 0>,-
0664 434 <<Quorum Disk Votes>, w_qdvotes, uw>,-
0664 435 <<Nodes>, w_nodes, uw>,-
0664 436 <<Quorum Disk>, t_qdname, ac, 15, 18>,-
0664 437 <<Found Node SYSID>, club_6bytes, club$b_fsystid>,-
0664 438 <<Founding Time>, date_routine, club$q_ftime>,-
0664 439 <<>, time_routine, club$q_ftime>,-
0664 440 <<Index of next CSID>, w_next_csid, xw>,-
0664 441 <<Quorum Disk Cntrl Block>, l_cludcb, xl, 25, 8>,-
0664 442 <<Timer Entry Address>, l_tqe, xl>,-
0664 443 <<CSP Queue>, l_cspfl, q2>,-
0664 444 <<Transaction code>, trans_byte, club$b_cur_code>,-
0664 445 <<Transaction Phase>, trans_byte, club$b_cur_phase>,-
0664 446 <<Message Count>, trans_word, club$w_msgcnt>,-
0664 447 >
0754 448
0754 449
0754 450 club_col_2:
0754 451 column_list
0754 452 club$, 21, 12, 0, <-
0754 453 <<Last transaction code>, b_lst_code, xb>,-
0754 454 <<Last trans. number>, l_lst_xtn, ul>,-
0754 455 <<Last coordinator CSID>, l_lst_coord, xl>,-
0754 456 <<Last time stamp>, date_routine, club$q_lst_time>,-
0754 457 <<>, time_routine, club$q_lst_time>,-
0754 458 <<Largest trans. id>, l_max_xtn, xl>,-
0754 459 <<Resource Alloc. retry>, l_retrycnt, ul>,-
0754 460 <<Figure of Merit>, l_fmerit, xl>,-
0754 461 <<Member State Seq. Num.>, w_memseq, xw>,-
0754 462 <<Foreign Cluster>, l_foreign_cluster, xl>,-
0754 463 <<Curr. coord. CSID>, trans_long, club$l_cur_coord>,-
0754 464 <<Current trans. number>, trans_long, club$l_cur_xtn>,-
0754 465 <<Curr. time-stamp>, curr_date, club$q_cur_time>,-
0754 466 <<>, curr_time, club$q_cur_time>,-
0754 467 >
0844 468
0844 469 :
0844 470 : The following are all PRINT_COLUMNS action routines for the show
0844 471 : cluster block displays.
0844 472 :
0844 473 : Action Routine Inputs:
0844 474 :
0844 475 : R2 value from the COLUMN_LIST entry
```



```
0844 476 : R5 size of value section for this item
0844 477 : R7 address of a descriptor for a scratch string in
0844 478 : which the FAO converted value is to be returned
0844 479 : R11 base address of the local CLUB copy
0844 480 :
0844 481 : Action Routine Outputs:
0844 482 :
0844 483 : R0 status
0844 484 : lbs ==> use this entry
0844 485 : lbc ==> skip this entry
0844 486 : R1-R5 scratch
0844 487 : all other registers must be preserved
0844 488 :
0844 489 :*****
0844 490 quor_vote:
52 20 AB 3C 0844 491 movzwl club$w_quorum(r11),r2 ; display quorum value with
53 22 AB 3C 0848 492 movzwl club$w_votes(r11),r3 ; the value for votes
084C 493 $fao_s
084C 494 ctrstr = club_2words,- ; two values as requested
084C 495 outbuf=(r7),-
084C 496 outlen=(r7),-
084C 497 p1=r2,-
084C 498 p2=r3
05 085F 499 rsb
0860 500
0860 501 :*****
53 5B 52 C1 0860 502 club_6bytes:
55 55 0C C2 0860 503 addl3 r2,r11,r3 ; locate storage of interest
0864 504 subl #12,r5 ; get size of filler field
0867 505 $fao_s
0867 506 ctrstr=club_fao_6bytes,-
0867 507 outbuf=(r7),-
0867 508 outlen=(r7),-
0867 509 p1=r5,-
0867 510 p2=4(r3),-
0867 511 p3=(r3)
05 087D 512 rsb
087E 513
087E 514 :*****
53 5B 52 C1 087E 515 dateRoutine:
000006ED'EF 63 7D 0882 516 addl3 r2,r11,r3 ; locate area of interest
0882 517
0889 518 movq (r3),tim_buffer ; move into buffer
0893 519 alloc 11,r4 ; allocate space for date
0893 520 $asctim_s timadr=tim_buffer,-
0893 521 timbuf=(r4) ; convert value to ascii
52 54 D0 08A6 522 movl r4,r2 ; pass address of descriptor in r2
5E 14 C0 08A9 523 do_column_entry as ; display date
05 08B2 524 addl #20,sp ; clean up the stack
08B5 525 rsb
08B6 526
08B6 527 :*****
53 5B 52 C1 08B6 528 timeRoutine:
000006ED'EF 63 7D 08BA 529 addl3 r2,r11,r3 ; locate area of interest
08C1 530 movq (r3),tim_buffer ; move into buffer
08CB 531 alloc 24,r4 ; allocate space for date/time
08CB 532 $asctim_s timadr=tim_buffer,-
```



```
04 64 09 B0 08CB 533 timbuf=(r4) ; convert to ascii
    A4 0B C0 08DE 534 movw #9,(r4) ; only display time - adjust length accordingly
    52 54 D0 08E1 535 addl2 #11,4(r4) ; adjust address to point to time
    5E 20 C0 08E5 536 movl r4,r2 ;
    05 08E8 537 do_column_entry as ; display time
    08F1 538 addl2 #32,sp ; clean up the stack
    08F4 539 rsb
    08F5 540
    08F5 541 ;*****
    03 1C AB 1D E1 08F5 542 curr_date:
    08FA 543 b5c #club$transition,club$l_flags(r11),10$
    08FA 544 ; if transition in progress, this field is of
    81 AF 17 08FA 545 ; interest to us, so display.
    08FA 546 jmp date_routine
    08FD 547 10$:
    05 08FD 548 rsb
    08FE 549
    08FE 550 ;*****
    03 1C AB 1D E1 08FE 551 curr_time:
    0903 552 b5c #club$transition,club$l_flags(r11),10$
    0903 553 ; if transition in progress, this field is of
    B0 AF 17 0903 554 ; interest to us, so display
    0906 555 jmp time_routine
    05 0906 556 10$:
    0907 557 rsb
    0907 558
    0907 559 ;*****
    0C 1C AB 1D E1 0907 560 trans_long:
    090C 561 b5c #club$transition,club$l_flags(r11),10$
    090C 562 ; if transition in progress, this field is of
    52 5B C0 090C 563 ; interest to us, so display.
    090F 564 addl r11,r2 ; locate cell to return
    0918 565 do_column_entry xl,jmp
    05 0918 566 10$:
    0918 567 rsb
    0919 568
    0919 569 ;*****
    0C 1C AB 1D E1 0919 570 trans_word:
    091E 571 b5c #club$transition,club$l_flags(r11),10$
    091E 572 ; if transition in progress, this field is of
    52 5B C0 091E 573 ; interest to us, so display.
    0921 574 addl r11,r2 ; locate cell to return
    092A 575 do_column_entry uw,jmp
    05 092A 576 10$:
    092A 577 rsb
    092B 578
    092B 579 ;*****
    0C 1C AB 1D E1 092B 580 trans_byte:
    0930 581 b5c #club$transition,club$l_flags(r11),10$
    0930 582 ; if transition in progress, this field is of
    52 5B C0 0930 583 ; interest to us, so display.
    0933 584 addl r11,r2 ; locate cell to return
    093C 585 do_column_entry ub,jmp
    05 093C 586 10$:
    093C 587 rsb
```



```
093D 589 .sbtll display_clufcb --- display cluster failover control block(CLUFCB)
093D 590 :---
093D 591 :
093D 592 : display_clufcb
093D 593 :
093D 594 : This routine displays the cluster failover control block which is
093D 595 : a subblock of the club that is used to sequence failover actions
093D 596 : in a cluster
093D 597 :
093D 598 : Inputs:
093D 599 :
093D 600 : 4(ap) = actual address of cluster failover control block
093D 601 : 8(ap) = address of cluster fcb in local storage
093D 602 :
093D 603 : Outputs:
093D 604 : The Cluster failover control block is displayed.
093D 605 :
093D 606 :---
093D 607 display_clufcb:
093D 608 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
093F 609
54 08 AC DO 093F 610 ensure 20 ; make sure at least 20 lines on screen
04 AC DD 0957 611 movl 8(ap),r4 ; cluster fcb in local storage
095B 612 pushl 4(ap) ; actual address of cluster fcb
095E 613 print 1,<!_ --- Cluster Failover Control Block (CLUFCB) !XL --->
096B 614 skip 1
0974 615 alloc 80,r5 ; 80 byte output buffer
20 A4 DD 0986 616 pushl clufcb$l_status(r4) ; status
F793 CF 9F 0989 617 pushab fcb_status ; bit definition table
00000000'EF 02 FB 098D 618 calls #2,translate_bits ; translate bits to names
55 DD 0994 619 pushl r5 ; names defining status bits
20 A4 DD 0996 620 pushl clufcb$l_status(r4) ; status field
0999 621 print 2,<Flags: !XL !AS> ; display translated status
09A6 622 skip 1
5E 00000058 8F C0 09AF 623 addl2 #88,sp ; clean up stack
09B6 624 print_columns -
09B6 625 (r4),4(ap),-
09B6 626 fcb_col_1,fcb_col_2 ; display the cluster fcb
09CE 627 status
04 09D5 628 ret success
```



```
09D6 630 .sbttl display_cludcb --- display cluster quorum disk control block
09D6 631 :---
09D6 632 :
09D6 633 display_cludcb
09D6 634 :
09D6 635 :
09D6 636 Inputs:
09D6 637 :
09D6 638 4(ap) = actual address of cluster quorum disk control block
09D6 639 :
09D6 640 Outputs:
09D6 641 The Cluster quorum disk control block is displayed.
09D6 642 :
09D6 643 :---
09D6 644 display_cludcb:
OFFC 09D6 645 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
09D8 646 :
09D8 647 skip 3
55 04 AC DO 09E1 648 movl 4(ap),r5 ; actual address of cluster dcb
55 DD 09E5 649 pushl r5
09E7 650 print 1,<!-- Cluster Quorum Disk Control Block (CLUDCB) !XL --->
09F4 651 skip 1
54 000002FC'EF 9E 09FD 652 movab cludcb,r4
0A04 653 getmem (r5),(r4),#cludcb$length ; read into local storage
01 50 E8 0A15 654 blbs r0,20$ ; branch if able to read
05 0A18 655 rsb ; else, exit
0A19 656 20$:
0A19 657 alloc 80,r6 ; 80 byte output buffer
7E 20 A4 3C 0A2B 658 movzwl cludcb$w_state(r4),-(sp) ; status
F7C5 CF 9F 0A2F 659 pushab cludcb_sstate ; bit definition table
00000000'EF 02 FB 0A33 660 calls #2,translate_bits ; translate bits to names
56 DD 0A3A 661 pushl r6 ; names defining state bits
7E 20 A4 3C 0A3C 662 movzwl cludcb$w_state(r4),-(sp) ; state field
0A40 663 print 2,<State: !XW !AS> ; display translated state
66 50 8F 9A 0A4D 664 movzbl #80,(r6) ; reinitialize buffer
7E 22 A4 3C 0A51 665 movzwl cludcb$w_flags(r4),-(sp) ; translate flags now
F7CF CF 9F 0A55 666 pushab cludcb_flags ; bit definition table
00000000'EF 02 FB 0A59 667 calls #2,translate_bits ; translate to names
56 DD 0A60 668 pushl r6 ; names defining flags
7E 22 A4 3C 0A62 669 movzwl cludcb$w_flags(r4),-(sp) ; flags field
0A66 670 print 2,<Flags: !XW !AS> ; display translated flags
5E 00000058 8F C0 0A73 671 skip 1
0A7C 672 addl2 #88,sp ; clean up stack
0A83 673 print_columns -
0A83 674 (r4),r5,-
0A83 675 dcb_col_1,dcb_col_2 ; display the cluster fcb
0A9A 676 :
0A9A 677 status success
04 0AA1 678 ret
```



```

0AA2 680 .sbttl cluster failover control block tables & action routines
0AA2 681 :
0AA2 682 : PRINT_COLUMNS table for CLUFCB displays
0AA2 683 :
0AA2 684 fcb_col_1:
0AA2 685 column_list -
0AA2 686 clufcb$, 21, 12, 4, < -
0AA2 687 <<Failover Step Index>,l_step,xl>,-
0AA2 688 <<Failover Instance ID>,l_id,xl>,-
0AA2 689 >
0AD2 690
0AD2 691 fcb_col_2:
0AD2 692 column_list -
0AD2 693 clufcb$, 21, 12, 4, < -
0AD2 694 <<CSB of Synchr. System>,l_sync_csb,xl>,-
0AD2 695 >

```



```

OAF2 697 .sbttl cluster quorum disk control block tables & action routines
OAF2 698 :
OAF2 699 : PRINT_COLUMNS table for CLUDCB displays
OAF2 700 :
OAF2 701 dcb_col_1:
OAF2 702 column_list -
OAF2 703 cludcb$, 21, 12, 4, < -
OAF2 704 <<Iteration Counter>, b_counter, ub>,-
OAF2 705 <<Activity Counter>, l_act_count, ul>,-
OAF2 706 <<Quorum file LBN>, l_qflbn, xl>,-
OAF2 707 >
OB32 708
OB32 709 dcb_col_2:
OB32 710 column_list -
OB32 711 cludcb$, 21, 12, 0, < -
OB32 712 << UCB address>, l_ucb, xl>,-
OB32 713 << TQE address>, l_tqe, xl>,-
OB32 714 << IRP address>, l_irp, xl>,-
OB32 715 >

```



```
display_csb --- display cluster system block (CSB)

0B72 717 .sbttl
0B72 718 :---
0B72 719 :
0B72 720 : display_csb
0B72 721 :
0B72 722 :
0B72 723 : Inputs:
0B72 724 :
0B72 725 : 4(ap) = actual address of cluster system block
0B72 726 :
0B72 727 : Outputs:
0B72 728 : The Cluster system block is displayed.
0B72 729 : All registers are preserved.
0B72 730 :
0B72 731 :---
0B72 732 display_csb:
0B72 733 .word
0B74 734
0B74 735 ensure 20
04 AC DD 0B8C 736 pushl 4(ap) ; actual address of csb
54 00000250'EF 9E 0B8F 737 skip 1
0B98 738 movab csb,r4 ; buffer to hold contents of csb locally
0B9F 739 getmem 24(ap),(r4),#csb$length ; read into local storage
01 50 E8 0BB1 740 blbs r0,10$ ; branch if able to read
05 0BB4 741 rsb ; else, exit
0BB5 742 10$:
57 68 A4 00000044 8F C1 0BB5 743 addl3 #sb$t_nodename,csb$l_sb(r4),r7 ; address of system block
0BBE 744 getmem (r7),node,#sb$s_nodename ; read node name
000006A5'EF DF 0BCF 745 pushal node ; address of ASCII string
0BD5 746 print 1,<!-- !AC Cluster System Block (CSB) !XL --->
0BE2 747 skip 1
0BEB 748
0BEB 749 make_csb_symbols:
0C01 750 make_symbol CSB, 4(ap)
0C17 751 make_symbol CDT, csb$l_cdt(r4)
0C2D 752 make_symbol PDT, csb$l_pdt(r4)
0C43 753 make_symbol SB, csb$l_sb(r4)
52 43 A4 9A 0C43 754
53 F40D CF 9E 0C47 755 movzbl csb$b_state(r4),r2 ; state field
00000000'GF 16 0C4C 756 movab csb_states,r3 ; bit definition table
13 13 0C52 757 jsb g^translate_address ; translate bits to names
50 DD 0C54 758 beql notrans ; equal, unable to translate
7E 43 A4 9A 0C56 759 pushl r0 ; names defining state bits
0C5A 760 movzbl csb$b_state(r4),-(sp) ; state field
0C67 761 print 2,<State: !XB !AC> ; display translated state
0C67 762 notrans:
0C79 763 alloc 80,r6 ; allocate buffer
F438 CF 9F 0C7C 764 pushl csb$l_status(r4) ; translate status now
00000000'EF 02 FB 0C7C 765 pushab csb_status ; bit definition table
56 DD 0C80 766 calls #2,translate_bits ; translate to names
60 A4 DD 0C87 767 pushl r6 ; names defining status
0C89 768 pushl csb$l_status(r4) ; status field
0C8C 769 print 2,<Flags: !XL !AS> ; display translated status
0C99 770 skip 1
5E 00000058 8F C0 0CA2 771 addl2 #88,sp ; clean up stack
0CA9 772 print_columns -
0CA9 773 (r4),4(ap),-
```


CLUSTER
V04-000

```
SHOW CLUSTER INFORMATION
display_csb --- display cluster system b
04 0CA9 774
    OCA9 775
    OCC7 776
    OCC7 777
    OCCE 778
    OCCF 779
status
ret
csb_col_1,csb_col_2,-
csb_col_3
success
; display the csb
```

H 16

16-SEP-1984 01:24:07
5-SEP-1984 03:31:48

VAX/VMS Macro V04-00
[SDA.SRC]CLUSTER.MAR;1

Page 19
(13)


```
OCCF 781 .sbttl cluster system block tables & action routines
OCCF 782 :
OCCF 783 : PRINT_COLUMNS table for CSB displays
OCCF 784 :
OCCF 785
OCCF 786 csb_2words:
OCCF 787 string <!6UW/!UW>
OCDF 788
OCDF 789 csb_2bytes:
OCDF 790 string <!5UB/!UB>
OCEF 791
OCEF 792 csb_col_1:
OCEF 793 column_list -
OCEF 794 csb$, 14, 8, 4, < -
OCEF 795 <<Quorum/Votes>,csbquor_votes,0>,-
OCEF 796 <<Quor. Disk Vote>,w_qdvotes,uw,15,7>,-
OCEF 797 <<CSID>,l_csid,xl>,-
OCEF 798 <<Eco/Version>,eco_vers,0>,-
OCEF 799 <<Reconn. time>,l_timeout,xl>,-
OCEF 800 <<Ref. count>,b_ref_cnt,ub>,-
OCEF 801 <<Ref. time>,date_routine,csb$q_reftime,11,11>,-
OCEF 802 <<>,time_routine,csb$q_reftime,11,11>,-
OCEF 803 >
OD7F 804
OD7F 805 csb_col_2:
OD7F 806 column_list -
OD7F 807 csb$, 16, 8, 4, < -
OD7F 808 <<Next seq. number>,w_sendseqnm,xw>,-
OD7F 809 <<Last seq num rcvd>,w_rcvdseqnm,xw,17,7>,-
OD7F 810 <<Last ack. seq num>,w_ackrseqnm,xw,17,7>,-
OD7F 811 <<Unacked messages>,b_unackedmsgs,ub>,-
OD7F 812 <<Ack limit>,b_reacktim,ub,18,6>,-
OD7F 813 <<Incarnation>,date_routine,csb$q_swincarn,13,11>,-
OD7F 814 <<>,time_routine,csb$q_swincarn,13,11>,-
OD7F 815 <<Lock mgr dir wgt>,w_lckdirwt,uw>,-
OD7F 816 >
OE0F 817
OE0F 818 csb_col_3:
OE0F 819 column_list -
OE0F 820 csb$, 16, 8, 0, < -
OE0F 821 <<Send queue>,l_sentqfl,xl>,-
OE0F 822 <<Resend queue>,l_resendqfl,xl>,-
OE0F 823 <<Block xfer Q.>,l_partnerqfl,q2>,-
OE0F 824 <<CDT address>,l_cdt,xl>,-
OE0F 825 <<PDT address>,l_pdt,xl>,-
OE0F 826 <<TQE address>,l_tqe,xl>,-
OE0F 827 <<SB address>,l_sb,xl>,-
OE0F 828 <<Current CDRP>,l_currcdrp,xl>,-
OE0F 829 >
OE9F 830 :
OE9F 831 : The following are all PRINT_COLUMNS action routines for the show
OE9F 832 : cluster block displays.
OE9F 833 :
OE9F 834 : Action Routine Inputs:
OE9F 835 :
OE9F 836 : R2 value from the COLUMN_LIST entry
OE9F 837 : R5 size of value section for this item
```



```
OE9F 838 : R7 address of a descriptor for a scratch string in
OE9F 839 : which the FAO converted value is to be returned
OE9F 840 : R11 base address of the local CLUB copy
OE9F 841 :
OE9F 842 : Action Routine Outputs:
OE9F 843 :
OE9F 844 : R0 status
OE9F 845 : lbs ==> use this entry
OE9F 846 : lbc ==> skip this entry
OE9F 847 : R1-R5 scratch
OE9F 848 : all other registers must be preserved
OE9F 849 :
OE9F 850 :
OE9F 851 :*****
OE9F 852 :csbquor_votes:
52 52 AB 3C OE9F 853 movzwl csb$w_quorum(r11),r2 ; display quorum value with
53 50 AB 3C OEA3 854 movzwl csb$w_votes(r11),r3 ; the value for votes
OEA7 855 $fao_s -
OEA7 856 ctrstr = csb_2words,- ; two values as requested
OEA7 857 outbuf=(r7),-
OEA7 858 outlen=(r7),-
OEA7 859 p1=r2,-
OEA7 860 p2=r3
05 OEBA 861 rsb
OE9F 862 :*****
52 40 AB 9A OE9F 863 eco_vers:
53 41 AB 9A OE9F 864 movzbl csb$b_ecolvl(r11),r2 ; display eco level with
OEBF 865 movzbl csb$b_vernum(r11),r3 ; the version number
OEC3 866 $fao_s -
OEC3 867 ctrstr = csb_2bytes,- ; two values as requested
OEC3 868 outbuf=(r7),-
OEC3 869 outlen=(r7),-
OEC3 870 p1=r2,-
OEC3 871 p2=r3
05 OED6 872 rsb
OED7 873
```



```
OED7 875 .sbttl show_scs --- display system communications(SCS) data structures
OED7 876 :---
OED7 877 :
OED7 878 : show_scs
OED7 879 :
OED7 880 : This is the main routine whose purpose is to provide information
OED7 881 : on vaxclusters which are related to system communications (SCS).
OED7 882 : Several structures are displayed. The order
OED7 883 : is as follows:
OED7 884 : summary page
OED7 885 : each system block and all of its path blocks
OED7 886 :
OED7 887 : Inputs:
OED7 888 :
OED7 889 : AP = pointer to TPARSE block
OED7 890 :
OED7 891 : Outputs:
OED7 892 :
OED7 893 : SCS data structures ( as mentioned above) are shown
OED7 894 : All registers are preserved.
OED7 895 :
OED7 896 :---
OED7 897 .enable lsb
OED7 898 show_scs::
OFFC OED7 899 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
OED9 900
OED9 901 subhd <VAXcluster data structures> ; set heading
OEE6 902 skip page
00000F06'EF 00 FB OEE6 903 calls #0,scs_summary ; summary page
OEF4 904
OEF4 905 skip page
00001091'EF 00 FB OEF4 906 calls #0,display_sb_pbs ; system and path blocks
OF02 907
50 01 D0 OF02 908 movl #1,r0 ; return success
04 OF05 909 ret
OF06 910 .dsabl lsb
```



```
OF06 912 .sbtll scs_summary --- display system communications(SCS) summary
OF06 913 :---
OF06 914 :
OF06 915 : show_scs
OF06 916 :
OF06 917 : This is a coroutine whose purpose is display a summary page.
OF06 918 : The summary page is divided into 2 parts. The first half
OF06 919 : displays a list of the local processes that are known to SCS.
OF06 920 : The second half displays a brief description of the systems
OF06 921 : in the cluster and the number of paths each has.
OF06 922 :
OF06 923 : Inputs:
OF06 924 :
OF06 925 : none
OF06 926 :
OF06 927 : Outputs:
OF06 928 :
OF06 929 : SCS data structures (as mentioned above) are shown
OF06 930 : All registers are preserved.
OF06 931 :
OF06 932 :---
OF06 933 :.enabl lsb
OF06 934 :scs_summary:
OF06 935 :.word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
OF08 936 :
OF08 937 : skip 1
OF11 938 : print 0,<!-- SCS Listening Process Directory --->
OF1E 939 : skip 1
OF27 940 : print 0,<Entry Address Connection ID Process Name Informati
OF34 941 : print 0,<-----
OF41 942 : skip 1
OF4A 943 :
OF4A 944 : getmem @scs$gq_direct,r6 ; get address of 1st entry
OF5A 945 : retiferr ; check for error
54 00000609'EF 9E OF5E 946 : movab directory,r4 ; local storage for entry
OF65 947 : getmem (r6),(r4),#sdir$c_length ; read into local storage
OF72 948 : retiferr ; check for error
OF76 949 5$:
1C A4 DF OF76 950 : pushal sdir$b_procinfr4 ; process information
1C A4 95 OF79 951 : tstb sdir$b_procinfr4 ; check for ascii info
04 13 OF7C 952 : beql 7$ ; equal, don't display
10 DD OF7E 953 : pushl #16
02 11 OF80 954 : brb 8$ ; branch around
00 DD OF82 955 7$: : don't display info
0C A4 DF OF84 956 8$: : process name
10 DD OF87 957 : pushal sdir$b_procnamr4
2C A4 DD OF89 958 : pushl #16
56 DD OF8C 959 : pushl sdir$l_conidr4 ; connection id
OF8E 960 : pushl r6 ; address of the dir entry
00000000'EF 64 D1 OF9B 961 : print 4,< !XL !XL !AD !AD>
13 13 OFA2 962 : cmpl sdir$l_flinkr4,scs$gq_direct ; any more entries
56 64 D0 OFA4 963 : beql 10$ ; equal, no
OFA7 964 : movl sdir$l_flinkr4,r6 ; get next entry address
BF 50 E8 OFB4 965 : getmem (r6),(r4),#sdir$c_length ; read into local storage
OFB7 966 : blbs r0,5$ ; branch if read o.k.
OFB7 967 :
OFB7 968 : This is the second half of the summary page, a brief description of the system
: blocks and the number of paths each system has.
```



```

OFB7 969 :
OFB7 970 10$:
OFB7 971 skip page
OFBE 972 skip 1
OFC7 973 print 0,< !_!_ --- SCS Systems Summary --->
OFD4 974 skip 1
OFDD 975 print 0,< SB Address Node Type System ID Paths>
OFEA 976 print 0,< ----->
OFF7 977 skip 1
1000 978
1000 979 getmem @scs$gq_config,r7 ; system block address
1010 980 retiferr
58 00000645'EF 9E 1014 981 movab sblock,r8 ; local storage for sb
101B 982 15$:
101B 983 getmem (r7),(r8),#sb$c_length ; read into local storage
102C 984 retiferr
1030 985
39 10 1030 986 bsb count_paths ; determine number of pb
54 DD 1032 987 pushl r4 ; move count onto stack
18 A8 DD 1034 988 pushl sb$b_systemid(r8) ; system id
7E 1C A8 3C 1037 989 movzwl sb$b_systemid+4(r8),-(sp) ; high order 2 bytes
24 A8 DF 103B 990 pushal sb$t_swtype(r8) ; type of node
24 A8 95 103E 991 tstb sb$t_swtype(r8) ; check for missing type
24 13 1041 992 beql 25$ ; equal, missing type
04 DD 1043 993 pushl #4 ;
44 A8 9F 1045 994 16$: pushab sb$t_nodename(r8) ; node name
57 DD 1048 995 pushl r7 ; address of sb
104A 996 print 6,< !XL !6<!AC!> !4<!AD!> !XL!XW !UL>
1057 997
57 68 D0 1057 998 movl sb$l_flink(r8),r7 ; move to next sb
00000000'EF 57 D1 105A 999 cmpl r7,scs$gq_config ; end of list of sb's
B8 12 1061 1000 bneq 15$ ; branch if no
1063 1001
1063 1002 ; We are done so let's get out of here.
1063 1003
1063 1004 20$:
50 01 D0 1063 1005 movl #1,r0 ; return success
04 1066 1006 ret
00 DD 1067 1007
DA 11 1067 1008 25$: pushl #0 ; put zero length on stack for type
1069 1009 brb 16$ ; continue
106B 1010 .dsabl lsb
106B 1011
106B 1012 Count_paths:
55 57 54 D4 106B 1013 clrl r4 ; counter for number of paths
55 57 0C C1 106D 1014 addl3 #sb$l_pbfl,r7,r5 ; start of list of pb's
55 0C A8 D1 1071 1015 cmpl sb$l_pbfl(r8),r5 ; any path blocks?
19 13 1075 1016 beql 35$ ; equal, zero pb's
56 0C A8 D0 1077 1017 movl sb$l_pbfl(r8),r6 ; get next path block
25$:
54 96 107B 1018 incb r4 ; increment count
107D 1020 assume pb$l_flink eq 0
107D 1021 getmem (r6) ; read in link to next pb
55 51 D1 1086 1022 cmpl r1,r5 ; end of list? (assume statement for r1)
05 13 1089 1023 beql 35$ ; equal, yes, so return to caller
56 51 D0 108B 1024 movl r1,r6 ; follow the link (assume statement for r1)
EB 11 108E 1025 brb 25$ ; still in loop of pb's
```


CLUSTER
V04-000

SHOW CLUSTER INFORMATION
scs_summary --- display system communica

B 1

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 25
(16)

05 1090 1026
1090 1027 35\$: rsb

; return to main line code


```
1091 1029 .sbtll display_sb_pbs --- display all system and path blocks
1091 1030 ---
1091 1031
1091 1032 display_sb_pbs
1091 1033
1091 1034 This is a coroutine whose purpose is display each system
1091 1035 block and its associated path blocks.
1091 1036
1091 1037 Inputs:
1091 1038
1091 1039 none
1091 1040
1091 1041 Outputs:
1091 1042
1091 1043 SCS data structures ( as mentioned above) are shown
1091 1044 All registers are preserved.
1091 1045
1091 1046 ---
1091 1047 display_sb_pbs:
OFFC 1091 1048 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
1093 1049
1093 1050 getmem @scs$gq_config,r3 ; system block address
10A3 1051 retiferr ; return if error
10A7 1052 10$:
00000000'EF 53 DD 10A7 1053 pushl r3 ; pass sb address
01 01 FB 10A9 1054 calls #1,show_system_block ; easy way out!
10B0 1055 assume sb$l_flink eq 0
10B0 1056 getmem (r3),r3 ; get next address
00000000'EF 53 D1 10BC 1057 cmpl r3,scs$gq_config ; are we at the end?
E2 12 10C3 1058 bneq 10$ ; not equal, still more
50 01 D0 10C5 1059
04 04 10C5 1060 movl #1,r0 ; return success
10C8 1061 ret
```



```
10C9 1063 .sbttl show_connections --- display all connection descriptor tables (CDT)
10C9 1064 ---
10C9 1065
10C9 1066 show_connections
10C9 1067
10C9 1068 This is the main routine whose purpose is to display the contents
10C9 1069 of each connection descriptor table(CDT). A CDT is used to store
10C9 1070 information about a virtual circuit between two processes. The
10C9 1071 first page is a brief summary of each cdt.
10C9 1072
10C9 1073 Inputs:
10C9 1074
10C9 1075 AP = pointer to TPARSE block
10C9 1076
10C9 1077 Outputs:
10C9 1078
10C9 1079 SCS data structures ( as mentioned above) are shown
10C9 1080 All registers are preserved.
10C9 1081
10C9 1082 ---
10C9 1083 .enabl lsb
00 10C9 1084 null_string:
10C9 1085 .byte 0
10CA 1086 .ascii //
10CA 1087
OFFC 10CA 1088 show_connections::
10CC 1089 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
10CC 1090
10CC 1091 Header information
10CC 1092
10CC 1093 subhd <VAXcluster data structures> ; set heading
10D9 1094 skip page
10E0 1095 skip 1
10E9 1096 print 0,<!_!_ --- CDT Summary Page --->
10F6 1097 skip 1
10FF 1098 print 0,<CDT Address Local Process Connection ID State Rem
110C 1099 print 0,<-----
1119 1100 skip 1
1122 1101
1122 1102 Now set up the data structures. Read the cdl ( specifically the location
1122 1103 containing the first free cdt and the list of cdt). Then read into local
1122 1104 storage the first cdt to display. Check to see if this cdt is on the free
1122 1105 list, if it is then it will not be displayed. Also if the state of the cdt
1122 1106 is closed, it will not be displayed. Otherwise it will be displayed.
1122 1107
1122 1108 getmem @scs$gl_cdl,r6 ; address of cdl
00000004'EF D4 1132 1109 clrl cdl_size ; initialize field
1138 1110 getmem cdl$w_size(r6),cdl_size,#2 ; size of cdl to read into virtual memory
114A 1111 retiferr ; return on error
00000000'EF 9F 114E 1112 pushab cdl ; will contain virtual address for cdl
00000004'EF 9F 1154 1113 pushab cdl_size ; size of cdl
00000000'GF 02 FB 115A 1114 calls #2,g^lib$get_vm ; get memory for cdl
1161 1115 retiferr ; return on error
1165 1116 getmem -16(r6),@cdl,cdl_size ; read cdl into storage
117B 1117 retiferr ; return on error
56 00000000'EF 00000004'EF C1 117F 1118 addl3 cdl_size,cdl,r6 ; r6 => end address of cdl
55 00000000'EF 10 C1 118B 1119 addl3 #16,cdl,r5 ; base of cdl
```



```
5A  F0 A5  D0 1193 1120      movl    cdl$w_maxconidx(r5),r10 ; max number of cdt's in table
      3B 19 1197 1121      blss     20$ ; no cdt's in table
57  55  D0 1199 1122      movl    r5,r7 ; save address of cdl
      5B D4 119C 1123
56  57  D1 119E 1124      clrl     r11 ; counter of the number of free cdt's
      31 13 11A1 1125 5$:    cmpl     r7,r6 ; safety check for end of list
      00B2 30 11A3 1126      beql     20$ ; yes, exit loop
      05 50 E8 11A6 1127      bsbw     free_cdt_list ; check to see if on free cdt list
      F2 5A F4 11A9 1128      blbs     r0,10$ ; set, not on free list so display
      26 11 11AC 1129      sobgeq    r10,5$ ; on free list so don't display
59  00000008'EF 9E 11AE 1130      brb     20$ ; hit end of cdt's
      5A DD 11B5 1131 10$:    movab    cdt,r9 ; local storage for cdt
      00001281'EF DD 11C6 1132      getmem    a(r7)+,(r9),#cdt$c_length ; read into local storage
      5A 16 11C8 1133      pushl    r10 ; save r10
      CA 8E D0 11CE 1134      jsb     display_sumline ; display one line of the summary page
      5A 11 11D1 1135      movl     (sp)+,r10 ; restore r10
      5A F4 11D4 1136      sobgeq    r10,5$ ; get next cdt
      11D4 1137 :
      11D4 1138 : At this stage the summary page is almost complete. The last thing to display
      11D4 1139 : is the number of free cdt's. R11 contains this number.
      11D4 1140 :
      11D4 1141 20$:    skip     1
      5B DD 11DD 1142      pushl    r11
      11DF 1143      print     1,<Number of free CDT's: !UL>
      11EC 1144 :
      11EC 1145 : The summary page is done. Now we will again loop through all the
      11EC 1146 : cdt's in use and call upon a routine to display the full contents
      11EC 1147 : of each and every one of them.
      11EC 1148 :
54  F4 A5  D0 11EC 1149      movl     cdl$freecdt(r5),r4 ; first free cdt
5A  F0 A5  D0 11F0 1150      movl     cdl$w_maxconidx(r5),r10 ; max number of cdt's in table
56  55  D1 11F4 1151 30$:    cmpl     r5,r6 ; test for end of list
      48 13 11F7 1152      beql     40$ ; equal, end of list
54  65  D1 11F9 1153      cmpl     (r5),r4 ; beginning of free cdt's
      2D 13 11FC 1154      beql     35$ ; equal, update r6 to next free cdt
      11FE 1155      skip     page ; new page for each cdt
      1205 1156 :
      1205 1157 : First check the state to see if this cdt is marked as closed. If it is
      1205 1158 : don't bother to display.
      1205 1159 :
57  65  28  C1 1205 1160      addl3    #cdt$w_state,(r5),r7 ; point to cdt state
      1209 1161      getmem    (r7),r7 ; read in state
      1215 1162      retiferr ; return on error
      00 57 B1 1219 1163      cmpw     r7,#cdt$c_closed ; closed cdt?
      1E 13 121C 1164      beql     38$ ; equal, disregard closed cdt
      121E 1165 :
      000013A9'EF 7E 85 D0 121E 1166      movl     (r5)+,-(sp) ; address of cdt
      C9 5A F4 1221 1167      calls    #1,display_cdt ; display
      BA 5A F4 1228 1168      sobgeq    r10,30$ ; get next one
      05 11 122B 1169 35$:    getmem    a(r5)+,r4 ; get next free cdt in r6
      BA 5A F4 1237 1170      sobgeq    r10,30$ ; loop for next cdt
      05 11 123A 1171      brb     40$ ; end of list
      123C 1172 :
      B3 85 D5 123C 1173 38$:    tstl     (r5)+ ; increment r5
      5A F4 123E 1174      sobgeq    r10,30$ ; loop for next valid cdt
      1241 1175 :
      1241 1176 40$:
```



```
00000000'EF 9F 1241 1177      pushab cdl      ; address of virtual memory to deall.
00000004'EF 9F 1247 1178      pushab cdl_size
00000000'GF 02 FB 124D 1179      calls #2,lib$free_vm ; deallocate virtual memory
50 01 D0 1254 1180      movl #1,r0      ; success
04 1257 1181      ret      ; finished!!!
      1258 1182      .dsabl lsb
      1258 1183
      1258 1184
      1258 1185 free_cdt_list:
54 50 01 D0 1258 1186      movl #1,r0      ; assume not on the cdt free list
54 F4 A5 D0 125B 1187      movl cd($l_freecdt(r5),r4 ; head of list
54 67 D1 125F 1188 5$:      cmpl (r7),r4      ; on the free list
      16 13 1262 1189      beql 10$      ; equal, on free list
      1264 1190      getmem (r4),r4      ; chain down the list
      1270 1191      retiferr      ; return on error
      1274 1192      tstl r4      ; end of list
      08 13 1276 1193      beql 15$      ; yes end of list and no match
      E5 11 1278 1194      brb 5$      ; loop and compare
      87 D5 127A 1195 10$:      tstl (r7)+      ; point to next cdt in list
      5B D6 127C 1196      incl r11      ; increment the counter of free cdt's
      50 D4 127E 1197      clrl r0      ; r0 clr indicates free list
      05 1280 1198 15$:      rsb      ; return to caller
      1281 1199
```



```
1281 1201 .sbttl      display_sumline --- display a line of the cdt summary page
1281 1202 :---
1281 1203 :
1281 1204 :      display_sumline
1281 1205 :
1281 1206 :      This is the subroutine whose purpose is to display a line of the
1281 1207 :      summary page for the given cdt.
1281 1208 :
1281 1209 :      Inputs:
1281 1210 :
1281 1211 :          R9 = address of cdt in local storage
1281 1212 :
1281 1213 :      Outputs:
1281 1214 :
1281 1215 :          A line of the cdt summary page is displayed.
1281 1216 :
1281 1217 :---
1281 1218 :      .enabl  lsb
1281 1219 :      display_sumline:
00  28 A9  B1 1281 1220 :      cmpw    cdt$w_state(r9),#cdt$c_closed ; if closed, ignore this cdt
      03  12 1285 1221 :      bneq    1$ ; if not equal, cdt is of interest
      003F 31 1287 1222 :      brw     40$ ; neq, not closed so continue to process
128A 1223 :
128A 1224 :      Now obtain the remote node name. This requires going through a few
128A 1225 :      channels. The CDT contains the path block address. The path block
128A 1226 :      will lead us to the system block which yields the remote node name.
128A 1227 :      Follow me. However if the cdt is a listen cdt, the remote node name
128A 1228 :      will not be present. So test first for this condition.
128A 1229 :
128A 1230 :      1$:      cmpw    cdt$w_state(r9),#cdt$c_listen ; listen cdt?
01  28 A9  B1 128E 1231 :      bneq    5$ ; no equal, not a listen cdt
      06  12 1290 1232 :      pushab  null_string ; listen cdt - no remote node name
FE35 CF 9F 1294 1233 :      brb     10$ ;
      0B  11 1296 1234 :      5$:      pushl   r9 ; address of cdt in local storage
0000136D'EF 59 DD 1296 1235 :      calls   #1,remote_node ; find remote node name
      01  FB 1298 1236 :      pushl   r10 ; address of counted ascii string
      5A  DD 129F 1237 :
12A1 1238 :
12A1 1239 :      Cdt's are not the easiest data structure to display. The key to
12A1 1240 :      displaying them correctly is the state and blkstate fields. The
12A1 1241 :      strategy is to translate the blkstate field first. If this succeeds,
12A1 1242 :      then displaying the cdt follows the normal path. If this translation
12A1 1243 :      fails, then the state field is translated. The blkst_flag will be
12A1 1244 :      clear if the translation of it succeeded. Setting of the flag indicates
12A1 1245 :      failure.
12A1 1246 :
12A1 1247 :      10$:     jsb     state_translate ; translate state value
000012CA'EF 16 DD 12A7 1248 :      pushl   r0 ; address of counted ascii string
      50  DD 12A9 1249 :      pushl   cdt$l_lconid(r9) ; connection id
      18 A9 DD 12AC 1250 :
12AC 1251 :      Now to display the local process name. This also is not as straight forward
12AC 1252 :      as it seems. If the cdt is NOT a listen CDT, then the process name in the
12AC 1253 :      cdt is valid. However if the cdt is a listen cdt, the local process name
12AC 1254 :      field will most likely be zero. (Bummer!) There is hope however for re-
12AC 1255 :      trieving the name. The cdt will have a valid connection id. Therefore,
12AC 1256 :      if search the scs directory looking for a match on connection id, the
12AC 1257 :      entry which matches based on the id will contain the local process name.
```


CLUSTER
V04-000

SHOW CLUSTER INFORMATION

H 1

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 31
(19)

```
000012EB'EF  59 DD 12AC 1258 ;
              01 FB 12AC 1259 ; address of cdt in local storage
              52 DD 12AE 1260 ; find the local process name
              53 DD 12B5 1261 ; address of process name
              FC A7 DD 12B7 1262 ; maximum length of name
              DD 12B9 1263 ; address of cdt
              12BC 1264 ; address of cdt
              05 12C9 1265 40$: print 6,< !XL !AD !XL !9<!AC!> !AC>
              12CA 1266 .rsb ; return to main routine
              .dsabl lsb
```



```
12CA 1268 .sbttl state_translate --- translate cdt state values to names
12CA 1269 ---
12CA 1270 :
12CA 1271 : state_translate
12CA 1272 :
12CA 1273 : This is the subroutine whose purpose is to translate the state fields
12CA 1274 : in the given cdt to their corresponding ascii names for display.
12CA 1275 :
12CA 1276 : Inputs:
12CA 1277 :
12CA 1278 : R9 = address of cdt in local storage
12CA 1279 :
12CA 1280 : Outputs:
12CA 1281 :
12CA 1282 : State fields in the cdt are translated to meaningful
12CA 1283 : ascii names.
12CA 1284 :
12CA 1285 : ---
12CA 1286 : state_translate:
12CA 1287 :
12CA 1288 : Translate the SCS blocked send state location to their equivalent
12CA 1289 : ascii names.
12CA 1290 :
12CA 1291 : movzwl cdt$w_blkstate(r9),r2 ; scs send blocked state
12CE 1292 : movab cdt_blkstate,r3 ; definition table
12D3 1293 : jsb g^ttranslate_address ; translate constants to names
12D9 1294 : bneq 10$ ; not equal, match found
12DB 1295 :
12DB 1296 : The translation has failed. Translate the state value and set the flag
12DB 1297 : to indicate that the blkstate translation has failed.
12DB 1298 :
12DB 1299 : movzwl cdt$w_state(r9),r2 ; state value
12DF 1300 : movab cdt_sstate,r3 ; corresponding definition table
12E4 1301 : jsb g^ttranslate_address ; translate
12EA 1302 10$:
05 12EA 1303 : rsb
```

52 2A A9 3C 12CA 1291
53 EFF6 CF 9E 12CE 1292
00000000'GF 16 12D3 1293
OF 12 12D9 1294

52 28 A9 3C 12DB 1299
53 EF75 CF 9E 12DF 1300
00000000'GF 16 12E4 1301
12EA 1302
05 12EA 1303


```
12EB 1305 .sbtll find_procname --- Find the local process name.
12EB 1306 ---
12EB 1307
12EB 1308 find_procname
12EB 1309
12EB 1310 This is the coroutine whose purpose is to find the local process
12EB 1311 name. If the cdt is a listen cdt, then the local process name in
12EB 1312 the cdt will not be valid. We will have to use the scs directory
12EB 1313 to determine the name. The connection id of the cdt is used to find
12EB 1314 the scs directory entry with the local process name for this cdt.
12EB 1315 If the state of the cdt is other than listen, then the field in
12EB 1316 the cdt should be valid.
12EB 1317
12EB 1318 Inputs:
12EB 1319
12EB 1320 4(AP) = address of cdt in local storage
12EB 1321
12EB 1322 Outputs:
12EB 1323
12EB 1324 R2 = address of the local process name.
12EB 1325 R3 = length of the name.
12EB 1326 All other registers are preserved.
12EB 1327
12EB 1328 ---
12EB 1329 find_procname::
12EB 1330 .word ^m<r4,r5>
12ED 1331
12ED 1332 movl 4(ap),r5 ; get the address of the cdt
12F1 1333 cmpw cdt$w_state(r5),#cdt$c_listen ; is this a listen cdt
12F5 1334 beql 15$ ; equal, yes
12F7 1335 tstl cdt$l_lprocnam(r5) ; test for zero address
12FA 1336 beql 10$ ; no local process name
12FC 1337 movab procname,r2 ; address of local process
1303 1338 getmem @cdt$l_lprocnam(r5),(r2),#16 ; read into local storage
1311 1339 movl #16,r3
1314 1340 brw 50$
1317 1341 10$: movab null_string,r2 ; no local process name
131C 1342 movl #0,r3 ; zero length
131F 1343 brw 50$
1322 1344
1322 1345 ; Search the scs directory for the entry with the same connection id.
1322 1346
1322 1347 15$: movab directory,r4 ; local storage for directory entry
1329 1348 getmem @scs$gq_direct,(r4),#sdir$c_length
133A 1349 retiferr ; return if error
133E 1350 20$: cmpl cdt$l_lconid(r5),sdir$l_conid(r4)
1343 1351 beql 30$ ; match
1345 1352 movl sdir$l_flink(r4),r3 ; next entry
1348 1353 getmem (r3),(r4),#sdir$c_length ; read into local storage
1355 1354 retiferr ; return on error
1359 1355 brb 20$
135B 1356
135B 1357 30$: tstb sdir$b_procnam(r4) ; check if name exists
135E 1358 bneq 40$ ; not equal, exists
1360 1359 movl #0,r3 ; zero length
1363 1360 brb 45$ ; branch around
1365 1361 40$: movl #16,r3 ; length
```


CLUSTER
V04-000

SHOW CLUSTER INFORMATION

K 1

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 34
(21)

52 0C A4 9E 1368 1362 45\$: movab sdir\$b_procnam(r4),r2 ; address of local process name
04 136C 1363 50\$: ret


```
136D 1365 .sbtll remote_node --- find the remote node name
136D 1366 ---
136D 1367 :
136D 1368 : remote_node
136D 1369 :
136D 1370 : This routine is to determine the name of the remote node
136D 1371 : given the contents of a cdt. The cdt will give us the path
136D 1372 : block address which in turn will yield the system block
136D 1373 : address. The system block contains the remote node name
136D 1374 : that is desired.
136D 1375 :
136D 1376 : Inputs:
136D 1377 :
136D 1378 : 4(ap) = address of cdt in local storage
136D 1379 :
136D 1380 : Outputs:
136D 1381 :
136D 1382 : R10 = address of counted ascii string of remote node name.
136D 1383 : All other registers are preserved.
136D 1384 :
136D 1385 : ---
136D 1386 : remote_node:
136D 1387 : .word ^m<r2,r3,r4,r5,r8,r9>
136F 1388 : movl 4(ap),r9 ; address of cdt in local storage
59 04 AC 033C 136F 1388 : movl cdt$l_pb(r9),r8 ; path block address
58 1C A9 D0 1373 1389 : addl2 #pb$l_sblink,r8 ; point to system block address
58 58 30 C0 1377 1390 : getmem (r8),r8 ; yields system block address
137A 1391 : retiferr ; return if error
1386 1392 : movab node,r10 ; local storage for node name
5A 000006A5'EF 9E 138A 1393 : addl2 #sb$t_nodename,r8 ; point at node name
58 00000044 8F C0 1391 1394 : getmem (r8),(r10),#sb$s_nodename ; read it into local storage
50 01 D0 13A5 1395 : movl #1,r0 ; success
04 13A8 1396 : ret ; return with address of node name in r10
13A9 1397
13A9 1398
```



```
13A9 1400 .sbttl display_cdt --- display a connection descriptor table
13A9 1401 :---
13A9 1402 :
13A9 1403 : display_cdt
13A9 1404 :
13A9 1405 : This is a coroutine whose purpose is display each connection
13A9 1406 : descriptor table (CDT). A CDT is used to store information
13A9 1407 : about a virtual circuit between two processes.
13A9 1408 :
13A9 1409 : Inputs:
13A9 1410 :
13A9 1411 : 4(ap) = actual address of cdt
13A9 1412 :
13A9 1413 : Outputs:
13A9 1414 :
13A9 1415 : CDT is displayed.
13A9 1416 : All registers are preserved.
13A9 1417 :
13A9 1418 :---
13A9 1419 : .enabl lsb
OFFC 13A9 1420 display_cdt::
13A9 1421 : .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
59 00000008'EF 9E 13AB 1422 :
13AB 1423 : movab cdt,r9 ; local storage for cdt
13B2 1424 : getmem @4(ap),(r9),#cdt$c_length ; read into local storage
13C4 1425 : retiferr
13C8 1426 : skip 1
04 AC DD 13D1 1427 : pushl 4(ap) ; actual address of cdt
13D4 1428 : print 0,<!-- Connection Descriptor Table (CDT) !XL --->
13E1 1429 : skip 1
13EA 1430 :
13EA 1431 : Determine local process name
13EA 1432 :
00 28 A9 B1 13EA 1433 : cmpw cdt$w_state(r9),#cdt$c_closed ; if closed, no local process name
0A 12 13EE 1434 : bneq 1$ ; not equal, continue to look
000006B5'EF 9F 13F0 1435 : pushab procname ; null process name
00 DD 13F6 1436 : pushl #0 ; zero length
21 11 13F8 1437 : brb 3$ ; no local process name available
54 A9 D5 13FA 1438 1$: tstl cdt$l_lprocnam(r9) ; check to see if valid address
03 12 13FD 1439 : bneq 2$ ; yes there is a process name
011E 31 13FF 1440 : brw 40$ ; go and search for it
1402 1441 :
52 000006B5'EF 9E 1402 1442 2$: movab procname,r2 ; put into local storage
1409 1443 : getmem @cdt$l_lprocnam(r9),(r2),#16 ; local process name
52 DD 1417 1444 : pushl r2 ; push address of process name
10 DD 1419 1445 : pushl #16 ; length of name
141B 1446 :
141B 1447 : Translate state value to ascii string
141B 1448 :
52 28 A9 3C 141B 1449 3$: movzwl cdt$w_state(r9),r2 ; state value
53 EE35 CF 9E 141F 1450 : movab cdt_sstate,r3 ; corresponding definition table
00000000'GF 16 1424 1451 : jsb @^translate_address ; translate
03 12 142A 1452 : bneq 4$ ; match found
00EA 31 142C 1453 : brw 35$ ; no match
50 DD 142F 1454 4$: pushl r0 ; translated to names
7E 28 A9 3C 1431 1455 5$: movzwl cdt$w_state(r9),-(sp) ; state value
1435 1456 : print 3,<State: !XW !AC Local Process: !AD>
```



```
1442 1457 ; display
1442 1458 :
1442 1459 : Determine remote process name if it exists.
1442 1460 :
00 28 A9 B1 1442 1461 cmpw cdt$w_state(r9),#cdt$sc_closed ; check for closed cdt
000006B5'EF 12 1446 1462 bneq 10$ ; not equal, continue to look for
00 00 DD 1448 1463 pushab procname ; null process name
27 11 144E 1464 pushl #0 ; zero length
50 A9 D5 1450 1465 brb 15$ ; no remote node and process available
22 13 1452 1466 10$: tstl cdt$l_rprocnam(r9) ; check for non-zero address
52 000006B5'EF 9E 1455 1467 beql 15$ ; equal, remote process name not available
1457 1468 movab procname,r2 ; local storage for remote process name
145E 1469 getmem @cdt$l_rprocnam(r9),(r2),#16 ; read into local storage
52 DD 146C 1470 pushl r2 ; address of remote process name
10 DD 146E 1471 pushl #16 ; length of name
1470 1472 :
1470 1473 : Obtain the remote node name.
1470 1474 :
FEF6 CF 59 DD 1470 1475 pushl r9 ; address of cdt in local storage
01 FB 1472 1476 calls #1,remote_node ; find remote node
5A DD 1477 1477 pushl r10 ; counted ascii string
1479 1478 :
1479 1479 : Translate scs blocked send state to ascii string
1479 1480 :
52 2A A9 3C 1479 1481 15$: movzwl cdt$w_blkstate(r9),r2 ; scs send blocked state
53 EE47 CF 9E 147D 1482 movab cdt_blkstate,r3 ; definition table
00000000'GF 16 1482 1483 jsb g^ttranslate_address ; translate constants to names
03 12 1488 1484 bneq 19$ ; translate failed if equal
0085 31 148A 1485 brw 30$
50 DD 148D 1486 19$: pushl r0 ; address of counted ascii string
7E 2A A9 3C 148F 1487 20$: movzwl cdt$w_blkstate(r9),-(sp) ; scs send blocked state
1493 1488 :
1493 1489 : Display
1493 1490 :
01 28 A9 B1 1493 1491 cmpw cdt$w_state(r9),#cdt$sc_listen ; check for listen cdt
15 13 1497 1492 beql 22$ ; equal, listen so no remote
00 28 A9 B1 1499 1493 cmpw cdt$w_state(r9),#cdt$sc_closed ; check for closed cdt
0F 13 149D 1494 beql 22$ ; no remote for closed
149F 1495 print 5,<Blocked State: !XW !AC Remote Node::Process: !AC::!A
0D 11 14AC 1496 brb 23$
14AE 1497 22$: print 2,<Blocked State: !XW !AC>
14BB 1498 :
14BB 1499 23$: make_symbol PB, cdt$l_pb(r9)
14D1 1500 make_symbol PDI, cdt$l_pdt(r9)
14E7 1501 skip 1
14F0 1502 :
14F0 1503 print_columns -
14F0 1504 (r9),4(ap),-
14F0 1505 cdt_col_1,cdt_col_2,cdt_col_3 ; display cdt
50 01 D0 150E 1506 25$: movl #1,r0 ; return with success
04 1511 1507 ret ; done
1512 1508 :
FBB3 CF 9F 1512 1509 30$: pushab null_string
FF76 31 1516 1510 brw 20$
1519 1511 :
FBAC CF 9F 1519 1512 35$: pushab null_string ; translation failed
FF11 31 151D 1513 brw 5$ ; return to main line code
```


CLUSTER
V04-000

SHOW CLUSTER INFORMATION

B 2

16-SEP-1984 01:24:07

VAX/VMS Macro V04-00

Page 38
(23)

display_cdt --- display a connection des

5-SEP-1984 03:31:48

[SDA.SRC]CLUSTER.MAR;1

FDC4	CF	59	DD	1520	1514				
		01	FB	1520	1515	40\$:	pushl	r9	; local address of cdt
		52	DD	1522	1516		calls	#1,find_procname	; maybe in the directory entry
		53	DD	1527	1517		pushl	r2	; address of local process name
		53	DD	1529	1518		pushl	r3	; length of name
	FEED	31		152B	1519		brw	3\$; return to main line code
				152E	1520				
				152E	1521		.dsabl	lsb	


```
152E 1523 .sbttl cdt_byaddr --- display the cdt requested by the user
152E 1524 :---
152E 1525 :
152E 1526 : cdt_byaddr
152E 1527 :
152E 1528 : This is a routine whose purpose is display a connection
152E 1529 : descriptor table (CDT) which the user has requested by using
152E 1530 : the /ADDR qualifier and a valid address of a cdt. A CDT is
152E 1531 : used to store information about a virtual circuit between
152E 1532 : two processes.
152E 1533 :
152E 1534 : Inputs:
152E 1535 :
152E 1536 : AP = TPARSE block (TPASL_NUMBER contains the address)
152E 1537 :
152E 1538 : Outputs:
152E 1539 :
152E 1540 : The requested CDT is displayed if a valid address is specified.
152E 1541 : Otherwise an informational message is sent to say invalid cdt
152E 1542 : address.
152E 1543 : All registers are preserved.
152E 1544 :
152E 1545 :---
152E 1546 :.enabl lsb
152E 1547 cdt_byaddr::
152E 1548 :.word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
1530 1549 :movl tpa$number(ap),r7 ; get address of cdt
1534 1550 :jsb verify_cdt ; is this a cdt
153A 1551 :blbc r0,10$ ; clear, not a cdt
153D 1552 :
153D 1553 : Now that we are through the validation phase the rest is trivial.
153D 1554 :
153D 1555 : skip page
1544 1556 :pushl r7 ; pass the actual address
1546 1557 :calls #1, display_cdt ; display it
154B 1558 10$:
154B 1559 :movl #1,r0
154E 1560 :ret
154F 1561
154F 1562 verify_cdt:
154F 1563 :tstl r7 ; check for 80000000 address
1551 1564 :bgeq 900$ ; not valid
1553 1565 :addl3 #cdt$b_type,r7,r8 ; point at the type field
1557 1566 :getmem (r8),r8 ; attempt to read
1563 1567 :retiferr ; return on error
1567 1568 :cmpb r8,#dyn$c_scs ; check for the right type
156B 1569 :bneq 900$ ; not equal, error
156D 1570 :ashl #-8,r8,r8 ; point at subtype
1572 1571 :cmpb r8,#dyn$c_scs_cdt ; check for correct subtype
1575 1572 :bneq 900$ ; not equal, invalid address
1577 1573 :movl #1,r0 ; valid cdt
157A 1574 :rsb
157B 1575
157B 1576 900$: : invalid cdt address
157D 1577 :pushl r7
157D 1577 :1,<!XL is not the address of a CDT>
15C5 1578 :movl #0,r0 ; invalid cdt
15C8 1579 :rsb
```

57 1C AC OFFC
0000154F'EF 16
OE 50 E9

FE5E CF 57 DD
01 FB
50 01 D0
04

58 57 0A C1

60 8F 58 91
OE 12
58 78
02 58 91
04 12
50 01 D0
05

58 58 F8 8F 78
02 58 91
04 12
50 01 D0
05

57 DD
00 D0
05

CLUSTER
V04-000

SHOW CLUSTER INFORMATION
cdt_byaddr --- display the cdt requested

D 2

16-SEP-1984 01:24:07
5-SEP-1984 03:31:48

VAX/VMS Macro V04-00
[SDA.SRC]CLUSTER.MAR;1

Page 40
(24)

15C9 1580
15C9 1581

.dsabl lsb


```
15C9 1583 .sbttl connection descriptor tables & action routines
15C9 1584 :
15C9 1585 : PRINT_COLUMNS table for CDT displays
15C9 1586 :
15C9 1587 :
15C9 1588 :
15C9 1589 cdt_col_1:
15C9 1590 column_list
15C9 1591 cdt$, 16, 8, 4, < -
15C9 1592 <<Local Con. ID>,l_lconid,xl>,-
15C9 1593 <<Remote Con. ID>,l_rconid,xl>,-
15C9 1594 <<Receive Credit>,w_rec,uw>,-
15C9 1595 <<Send Credit>,w_send,uw>,-
15C9 1596 <<Min. Rec. Credit>,w_minrec,uw>,-
15C9 1597 <<Pend Rec. Credit>,w_pendrec,uw>,-
15C9 1598 <<Initial Rec. Credit>,w_initlrec,uw,20,4>,-
15C9 1599 <<Rem. Sta.>,cdt_6bytes,cdt$b_rstation,12,12>,-
15C9 1600 <<Rej/Disconn Reason>,w_reason,uw,20,4>,-
15C9 1601 <<Queued for BDT>,w_qbdt_cnt,uw>,-
15C9 1602 <<Queued Send Credit>,w_qcr_cnt,uw,20,4>,-
15C9 1603 >
1689 1604 :
1689 1605 cdt_col_2:
1689 1606 column_list
1689 1607 cdt$, 16, 8, 4, < -
1689 1608 <<Datagrams sent>,l_dgsent,ul>,-
1689 1609 <<Datagrams rcvd>,l_dgrcvd,ul>,-
1689 1610 <<Datagram discard>,l_dgdiscard,ul>,-
1689 1611 <<Messages Sent>,l_msgsent,ul>,-
1689 1612 <<Messages Rcvd.>,l_msgrcvd,ul>,-
1689 1613 <<Send Data Init.>,l_snddats,ul>,-
1689 1614 <<Req Data Init.>,l_reqdats,ul>,-
1689 1615 <<Bytes Sent>,l_bytsent,ul>,-
1689 1616 <<Bytes rcvd>,l_bytreqd,ul>,-
1689 1617 <<Total bytes map>,l_bytmapd,ul>,-
1689 1618 >
1739 1619 :
1739 1620 cdt_col_3:
1739 1621 column_list
1739 1622 cdt$, 16, 8, 0, < -
1739 1623 <<Message queue>,l_waitqfl,q2>,-
1739 1624 <<Send Credit Q.>,l_crwaitqfl,q2>,-
1739 1625 <<PB address>,l_pb,xl>,-
1739 1626 <<PDT address>,l_pdt,xl>,-
1739 1627 <<Error Notify>,l_erraddr,xl>,-
1739 1628 <<Receive Buffer>,l_scsmsg,xl>,-
1739 1629 <<Connect Data>,l_condat,xl>,-
1739 1630 <<Aux. Structure>,l_auxstruc,xl>,-
1739 1631 >
17C9 1632 :
17C9 1633 : The following are all PRINT_COLUMNS action routines for the show
17C9 1634 : connection descriptor table displays.
17C9 1635 :
17C9 1636 : Action Routine Inputs:
17C9 1637 :
17C9 1638 : R2 value from the COLUMN_LIST entry
17C9 1639 : R5 size of value section for this item
```



```
17C9 1640 : R7 address of a descriptor for a scratch string in
17C9 1641 : which the FA0 converted value is to be returned
17C9 1642 : R11 base address of the local CDT copy
17C9 1643 :
17C9 1644 : Action Routine Outputs:
17C9 1645 :
17C9 1646 : R0 status
17C9 1647 : lbs ==> use this entry
17C9 1648 : lbc ==> skip this entry
17C9 1649 : R1-R5 scratch
17C9 1650 : all other registers must be preserved
17C9 1651 :
17C9 1652 cdt_fao_6bytes:
17C9 1653 string <!XW!XL>
17D7 1654
17D7 1655 :*****
17D7 1656 cdt_6bytes:
17D7 1657 addl3 r2,r11,r3 ; locate storage of interest
17DB 1658 subl #12,r5 ; get size of filler field
17DE 1659 $fao_s -
17DE 1660 ctrstr = cdt_fao_6bytes,-
17DE 1661 outbuf = (r7),-
17DE 1662 outlen = (r7),-
17DE 1663 p1 = r5,-
17DE 1664 p2 = (r3),-
17DE 1665 p3 = 4(r3)
05 17F3 1666 rsb
```

53 5B 52 C1
55 55 0C C2


```
17F4 1668 .sbttl show_rspid --- display RDT entries
17F4 1669 ----
17F4 1670 :
17F4 1671 : show_rspid
17F4 1672 :
17F4 1673 : This is the main routine whose purpose is to display the contents
17F4 1674 : of the response descriptor table(RDT). If an address of a connection
17F4 1675 : descriptor table is specified, then only those entries with the same
17F4 1676 : cdt will be displayed.
17F4 1677 :
17F4 1678 :
17F4 1679 : Inputs:
17F4 1680 :
17F4 1681 : AP = pointer to TPARSE block
17F4 1682 : CDT_SPCFY = 1 means /CONNECTION qualifier was present
17F4 1683 : 0 means the qualifier was absent
17F4 1684 :
17F4 1685 : Outputs:
17F4 1686 :
17F4 1687 : All RDT entries are displayed unless they are on the free list
17F4 1688 : or are permanently allocated in which cases they are of no
17F4 1689 : interest to us.
17F4 1690 : All registers are preserved.
17F4 1691 :
17F4 1692 :----
17F4 1693 : .enabl lsb
17F4 1694 :
17F4 1695 show_rspid::
OFFC 17F4 1696 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
17F4 1697 :
57 D4 17F6 1698 clrl r7 ; initialize cdt address
000006F9'EF D5 17F8 1699 tstl cdt_spcfy ; cdt address specified?
14 13 17FE 1700 beql 5$ ; equal, no address
57 1C AC D0 1800 1701 movl tpa$l_number(ap),r7 ; cdt address
000006F9'EF D4 1804 1702 clrl cdt_spcfy ; clear for next round
FD41 CF 16 180A 1703 jsb verify_cdt ; check validity of cdt
03 50 E8 180E 1704 blbs r0,5$ ; valid cdt
010D 31 1811 1705 brw 40$ ; not a cdt
1814 1706 :
1814 1707 : Header information
1814 1708 :
1814 1709 5$: subhd <VAXcluster data structures> ; set heading
1821 1710 skip page
1828 1711 skip 1
1831 1712 getmem @scs$gl_rdt,r6 ; address of rdt
1841 1713 retiferr
56 DD 1845 1714 pushl r6
1847 1715 print 1,<!-- Summary of Response Descriptor Table(RDT) !XL --->
1854 1716 skip 1
185D 1717 print 0,<RSPID CDRP Address CDT Address Local Process Name
186A 1718 print 0,<-----
1877 1719 skip 1
1880 1720 :
1880 1721 : Now set up the data structures. Read the rdt ( specifically the location
1880 1722 : containing the first free rd and the list of rds). Then read into local
1880 1723 : storage the first rd to display. Check to see if this rd is on the free
1880 1724 : list, if it is then it will not be displayed. Also if rd is permanently
```



```
1880 1725 : allocated, it will not be displayed. Otherwise it will be displayed.
1880 1726 :
0000063D'EF D4 1880 1727 clrl rdt_size ; initialize field
1886 1728 getmem rdt$w_size(r6),rdt_size,#2 ; size of rdt to read into virtual memory
1898 1729 retiferr ; return on error
00000639'EF 9F 189C 1730 pushab rdt ; will contain virtual address for rdt
0000063D'EF 9F 18A2 1731 pushab rdt_size ; size of rdt
00000000'GF 02 FB 18A8 1732 calls #2,g^lib$get_vm ; get memory for rdt
18AF 1733 retiferr ; return on error
18B3 1734 assume rdt$length eq 24
18B3 1735 getmem -24(r6),@rdt,rdt_size ; read rdt into storage
18C9 1736 retiferr ; return on error
59 00000639'EF 0000063D'EF C1 18CD 1737 addl3 rdt_size,rdt,r9 ; r9 => end address of rdt
55 00000639'EF 18 C1 18D9 1738 addl3 #rdt$length,rdt,r5 ; base of rdt
5A F8 A5 D0 18E1 1739 movl rdt$L_maxrdidx(r5),r10 ; max index of rdt
58 F4 A5 D0 18E5 1740 movl rdt$L_freerd(r5),r8 ; address of first free rdt entry
53 55 D0 18E9 1741 movl r5,r3 ; save base of rdt in r3
18EC 1742
54 D4 18EC 1743 clrl r4 ; initialize index counter to 0
59 55 D1 18EE 1744 10$: cmpl r5,r9 ; check for end of rd list
2E 13 18F1 1745 beql 40$ ; equal, end of list
58 65 D1 18F3 1746 cmpl (r5),r8 ; check to see if rd is free
1E 13 18F6 1747 beql 20$ ; equal, hit a free rd
1C 04 A5 00 E1 18F8 1748 bbc #rd$v_busy,4(r5),30$ ; not interested in perm. allocated rd's
57 D5 18FD 1749 tstl r7 ; cdt address specified
71 12 18FF 1750 bneq 50$ ; yes, so check for a match on cdt
54 DD 1901 1751 15$: pushl r4 ; index counter
55 DD 1903 1752 pushl r5 ; address of rd entry
00001995'EF 01 FB 1905 1753 calls #1,display_rd_entry ; display one line of the summary page
54 D6 190C 1754 incl r4 ; increment index counter
55 08 C0 190E 1755 addl2 #8,r5 ; advance to the next entry
DA 5A F4 1911 1756 sobgeq r10,10$ ; get next rd
08 11 1914 1757 brb 40$ ; end of list
58 65 D0 1916 1758 20$: movl (r5),r8 ; next free rd
54 D6 1919 1759 30$: incl r4 ; increment index counter
55 08 C0 191B 1760 addl2 #8,r5 ; advance to the next entry
CD 5A F4 191E 1761 sobgeq r10,10$ ; get next free rd
1921 1762 :
1921 1763 : Now that all the busy rdt entries have been displayed, let's walk through
1921 1764 : the wait queue and display its contents. These entries will not have a
1921 1765 : rspid since that is the reason they are in the wait queue.
1921 1766 :
56 56 FFFFFFFE8 8F C1 1921 1767 40$: addl3 #rdt$L_waitfl,r6,r6 ; start of wait queue
55 00000641'EF 9E 1929 1768 movab wait_cdrp,r5 ; address to hold cdrp
65 E8 A3 D0 1930 1769 movl rdt$L_waitfl(r3),(r5) ; first entry in wait queue
65 56 D1 1934 1770 41$: cmpl r6,(r5) ; end of queue?
22 13 1937 1771 beql 45$ ; yes, so exit
FFFFFFF 8F DD 1939 1772 pushl #-1 ; not available rspid
55 DD 193F 1773 pushl r5 ; address of cdrp address
00001995'EF 02 FB 1941 1774 calls #2,display_rd_entry ; display
1948 1775 getmem @r5,(r5),#4 ; get next entry
1955 1776 retiferr ; return on error
D9 11 1959 1777 brb 41$ ; loop
1958 1778 45$:
00000639'EF 9F 1958 1779 pushab rdt ;
0000063D'EF 9F 1961 1780 pushab rdt_size ; size to deallocate
00000000'GF 02 FB 1967 1781 calls #2,g^lib$free_vm ; deallocate virtual memory
```


CLUSTER
V04-000

SHOW CLUSTER INFORMATION
show_rspid --- display RDT entries

I 2

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 45
(26)

```
50 01 D0 196E 1782      movl    #1,r0      ; success status
      04 1971 1783      ret
      1972 1784
      1972 1785 50$:
58 58 65 D0 1972 1786      movl    (r5),r8      ; cdrp address
58 58 24 C1 1975 1787      addl3   #cdrp$l_cdt,r8,r8      ; point to cdt address
      1979 1788      getmem    (r8),r8      ; cdt address
57 58 D1 1985 1789      cmpl     r8,r7      ; match?
      8F 12 1988 1790      bneq    30$      ; not equal, don't display
      FF74 31 198A 1791      brw     15$      ; do display
      198D 1792
```



```
198D 1794 .sbtll display_rd_entry --- display an entry in the response descriptor tab
198D 1795 ---
198D 1796
198D 1797 display_rd_entry
198D 1798
198D 1799 This is a coroutine whose purpose is display each entry in the
198D 1800 response descriptor table (RDT). A RDT is used to provide a
198D 1801 match between a rspid and its CDRP.
198D 1802
198D 1803 Inputs:
198D 1804
198D 1805 4(ap) = rd entry in local storage
198D 1806 8(ap) = index portion of rspid
198D 1807
198D 1808 Outputs:
198D 1809
198D 1810 A rd entry is displayed.
198D 1811 All registers are preserved.
198D 1812
198D 1813 ---
198D 1814 .enabl lsb
198D 1815 no_rspid:
198D 1816 .ascic /waiting/
67 6E 69 74 69 61 77 00' 198D 1817 display_rd_entry::
07 198D 1818 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
OFFC 198D 1819
198D 1820 movl 4(ap),r4 ; address of rd in local storage
54 04 AC D0 198D 1821 movl (r4),r5 ; cdrp address
55 64 D0 198D 1822 addl3 #cdrp$l_cdt,r5,r6 ; cdt address
56 55 24 C1 198D 1823 getmem (r6),r6 ; get pointer to cdt from cdrp
198D 1824 tstl r6 ; test for valid cdt
56 D5 198D 1825 bgeq 10$ ; cdt is not valid
56 18 198D 1826 movab cdt,r7 ; address of local storage for cdt
57 00000008'EF 9E 198D 1827 getmem (r6),(r7),#cdt$c_length ; read into local storage
198D 1828 retiferr ; return on error
198D 1829 pushl r7 ; address of cdt in local storage
F998 CF 01 DD 198D 1830 calls #1,remote_node ; find the remote node
5A DD 198D 1831 pushl r10 ; counted ascii string
57 DD 198D 1832 pushl r7 ; address of cdt in local storage
F90D CF 01 FB 198D 1833 calls #1,find_procname ; find the local process name
52 DD 198D 1834 pushl r2 ; address of local process name
53 DD 198D 1835 pushl r3 ; length of name
56 DD 198D 1836 pushl r6 ; cdt address
55 DD 198D 1837 5$: pushl r5 ; cdrp address
08 AC D5 198D 1838 tstl 8(ap) ; rspid available
27 19 198D 1839 blss 20$ ; branch if not available
04 A4 00 B0 198D 1840 movw #0,4(r4) ; zero the state field
57 08 AC 04 A4 C9 198D 1841 bisl3 4(r4),8(ap),r7 ; yields the rspid
57 DD 198D 1842 pushl r7 ; rspid
198D 1843 print 6,<!XL !XL !XL !AD !AC>
50 01 D0 198D 1844 7$: movl #1,r0
04 198D 1845 ret
198D 1846
198D 1847 10$: pushab null_string ; empty string for node name
F6BD CF 9F 198D 1848 pushl #0 ; length of remote node name
00 DD 198D 1849 pushl #0 ; cdt address
00 DD 198D 1849
```


CLUSTER
V04-000

SHOW CLUSTER INFORMATION

K 2

16-SEP-1984 01:24:07
5-SEP-1984 03:31:48

VAX/VMS Macro V04-00
[SDA.SRC]CLUSTER.MAR;1

Page 47
(27)

D2	11	1A10	1850	brb	5\$			
		1A12	1851					
FF77	CF	9F	1A12	1852	20\$: pushab	no_rspid		
			1A16	1853	print	5,2 !AC	!XL	!XL
DF	11		1A23	1854	brb	7\$!AD>
			1A25	1855				
			1A25	1856	.dsabl	lsb		


```
show_ports --- display all port descriptor tables (PDT)

1A25 1858 .sbttl show_ports --- display all port descriptor tables (PDT)
1A25 1859 :---
1A25 1860 :
1A25 1861 : show_ports
1A25 1862 :
1A25 1863 : This is the main routine whose purpose is to display the contents
1A25 1864 : of each port descriptor table (PDT). A PDT is used to store
1A25 1865 : scs entry addresses and port independent bookkeeping. The first
1A25 1866 : page is a summary.
1A25 1867 :
1A25 1868 : Inputs:
1A25 1869 :
1A25 1870 : AP = pointer to TPARSE block
1A25 1871 :
1A25 1872 : Outputs:
1A25 1873 :
1A25 1874 : SCS data structures ( as mentioned above) are shown
1A25 1875 : All registers are preserved.
1A25 1876 :
1A25 1877 :---
1A25 1878 :.enabl lsb
OFFC 1A25 1879 show_ports::
1A25 1880 :.word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
1A27 1881 :
1A27 1882 : Header information
1A27 1883 :
1A27 1884 : subhd <VAXcluster data structures> ; set heading
1A34 1885 : skip page
1A3B 1886 : skip 1
1A44 1887 : print 0,<!_!_ --- PDT Summary Page --->
1A51 1888 : skip 1
1A5A 1889 : print 0,< PDT Address Type Device Driver Name>
1A67 1890 : print 0,< ----->
1A74 1891 : skip 1
1A7D 1892 :
1A7D 1893 : Read the contents of the pdt into local storage
1A7D 1894 :
54 00000525'EF 9E 1A7D 1895 : movab pdt,r4 ; local storage for pdt
1A84 1896 : getmem @scs$gl_pdt,r5 ; get address of 1st pdt
1A94 1897 : retiferr ; return on error
5B 55 D0 1A98 1898 : movl r5,r11 ; save the pdt address
1A9B 1899 10$: getmem (r5),(r4),#pdt$sc_length ; read pdt into local storage
1AAC 1900 : retiferr ; return on error
1AB0 1901 :
1AB0 1902 :
1AB0 1903 : Get the driver name from the device data block (DDB).
1AB0 1904 :
52 00DC C4 D0 1AB0 1905 : movl pdt$l_ucb0(r4),r2 ; ucb address
53 52 28 C1 1AB5 1906 : addl3 #ucb$l_ddb,r2,r3 ; point to the ddb
1AB9 1907 : getmem (r3),r6 ; ddb address
58 56 24 C1 1AC5 1908 : addl3 #ddb$b_drvnam_len,r6,r8 ; driver name length
1AC9 1909 : getmem (r8),r9 ; read it in
1AD5 1910 : retiferr ; return on error
59 59 9A 1AD9 1911 : movzbl r9,r9 ; zero the other fields
57 000006C5'EF 9E 1ADC 1912 : movab driver_name,r7 ; local storage for driver name
58 56 24 C1 1AE3 1913 : addl3 #ddb$t_drvname,r6,r8 ; point to driver name
1AE7 1914 : getmem 1(r8),(r7),r9 ; read into local storage
```



```
57 DD 1AF5 1915      retiferr      ; return on error
59 DD 1AF9 1916      pushl r7      ; address of driver name
      DD 1AFB 1917      pushl r9      ; length of driver name
      DD 1AFD 1918      :
      DD 1AFD 1919      : Put together the device name by pulling the unit number from the ucb and
      DD 1AFD 1920      : the device name from the device data block (DDB).
      DD 1AFD 1921      :
52 52 00000054 8F C1 1AFD 1922      addl3 #ucb$w_unit,r2,r2      ; point to the unit field
      DD 1B05 1923      getmem (r2),r2      ; unit number
      DD 1B11 1924      movzwl r2,-(sp)      ; put on the stack
58 56 14 C1 1B14 1925      addl3 #ddb$b_name_len,r6,r8      ; point to length field
      DD 1B18 1926      getmem (r8),r7      ; read the field
      DD 1B24 1927      retiferr      ; return on error
      DD 1B28 1928      movzbl r7,r7      ; zero the other fields
58 56 14 C1 1B2B 1929      addl3 #ddb$t_name,r6,r8      ; point to name field
56 000006D9'EF 9E 1B2F 1930      movab device_name,r6      ; local storage for name
      DD 1B36 1931      getmem 1(r8),(r6),r7      ; read the name
      DD 1B44 1932      retiferr      ; return on error
      DD 1B48 1933      pushl r6      ; address of name
57 DD 1B4A 1934      pushl r7      ; push length on stack
      DD 1B4C 1935      :
      DD 1B4C 1936      : Get the type of port (PA, PU, PE, PS).
      DD 1B4C 1937      :
      DD 1B4C 1938      movzbl pdt$b_pdt_type(r4),r2      ; port type
52 07 A4 9A 1B4C 1938      movab pdt_type,r3      ; translation table
53 E7AC CF 9E 1B50 1939      jsb g^translate_address      ; translate constant to name
      DD 1B55 1940      pushl r0      ; address of ascii name
      DD 1B5B 1941      pushl r5      ; pdt address
      DD 1B5D 1942      print 6,< !XL !AC !AD!UW !AD>
      DD 1B5F 1943      :
      DD 1B6C 1944      :
      DD 1B6C 1945      tstl pdt$l_flink(r4)      ; is there another pdt
      DD 1B6E 1946      beql 20$      ; equal, no
55 64 D0 1B70 1947      movl pdt$l_flink(r4),r5      ; next pdt to display
      DD 1B73 1948      brw 10$      ; loop for another pdt
      DD 1B76 1949      :
      DD 1B76 1950      : Now that the summary page is complete, let's go on to display each pdt
      DD 1B76 1951      : in full.
      DD 1B76 1952      :
      DD 1B76 1953      :
      DD 1B76 1954      20$: assume pdt$l_flink eq 0
      DD 1B78 1955      pushl r11      ; actual address of pdt
      DD 1B7F 1956      calls #2,display_pdt      ; display this pdt
      DD 1B8B 1957      getmem (r11),r11      ; read link field
      DD 1B8B 1957      retiferr      ; return on error
      DD 1B8F 1958      tstl r11      ; another pdt
      DD 1B91 1959      bneq 20$      ; not equal, display next pdt
      DD 1B93 1960      :
50 01 D0 1B93 1961      movl #1,r0      ; return with success
      DD 1B96 1962      ret
      DD 1B97 1963
```



```
1897 1965 .sbtll display_pdt --- display a port descriptor table
1897 1966 :---
1897 1967 :
1897 1968 : display_pdt
1897 1969 :
1897 1970 : This is a coroutine whose purpose is display each port
1897 1971 : descriptor table (PDT).
1897 1972 :
1897 1973 : Inputs:
1897 1974 :
1897 1975 : 4(ap) = actual address of pdt
1897 1976 :
1897 1977 : Outputs:
1897 1978 :
1897 1979 : PDT is displayed.
1897 1980 : All registers are preserved.
1897 1981 :
1897 1982 :---
1897 1983 .enabl lsb
OFFC 1897 1984 display_pdt::
1897 1985 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
1899 1986
1899 1987 skip page
1899 1988 skip 1
04 AC DD 18A9 1989 pushl 4(ap)
18AC 1990 print 0,<!_!_ --- Port Descriptor Table (PDT) !XL --->
18B9 1991 skip 1
55 00000525'EF DE 18C2 1992
18C2 1993 movl pdt,r5 ; local storage for pdt
18C9 1994 getmem @4(ap),(r5),#pdt$c_length ; read it in
18DB 1995 retiferr ; return on error
18DF 1996 :
18DF 1997 : Translate port type
18DF 1998 :
52 07 A5 9A 18DF 1999 movzbl pdt$b_pdt_type(r5),r2
53 E719 CF 9E 18E3 2000 movab pdt_type,r3
00000000'GF 16 18E8 2001 jsb g^translate_address ; get the ascii name for the
18EE 2002 ; port type
7E 07 A5 DD 18EE 2003 pushl r0 ; counted ascii string
9A 18F0 2004 movzbl pdt$b_pdt_type(r5),-(sp) ; port type
18F4 2005 print 2,<Type: !XB !AC>
1C01 2006 :
1C01 2007 : Translate port characteristics
1C01 2008 :
7E 04 A5 3C 1C10 2009 alloc 80 ; output buffer
E710 CF 9F 1C14 2010 movzwl pdt$w_portchar(r5),-(sp) ; port characteristics
00000000'EF 02 FB 1C18 2011 pushab port_char ; bit definition table
5E DD 1C1F 2012 calls #2,translate_bits ; translate bits to names
7E 04 A5 3C 1C21 2013 pushl sp ; address of string descriptor
5E 00000050 8F C0 1C25 2014 movzwl pdt$w_portchar(r5),-(sp) ; port characteristics
1C32 2015 print 2,<Characteristics: !XW !AS>
1C39 2016 addl2 #80,sp ; clean up the stack
1C42 2017 skip 1
1C42 2018 :
1C42 2019 : Display the rest of the pdt
1C42 2020 :
1C42 2021 make_symbol UCB, pdt$l_ucb0(r5)
```


CLUSTER
V04-000

SHOW CLUSTER INFORMATION

B 3

16-SEP-1984 01:24:07

VAX/VMS Macro V04-00

Page 51
(29)

display_pdt --- display a port descripto

5-SEP-1984 03:31:48

[SDA.SRC]CLUSTER.MAR;1

50	01	D0	1C59	2022	make_symbol	ADP, pdt\$l_adp(r5)	
		04	1C70	2023	print_columns	-	
			1C70	2024		(r5),4(ap),-	
			1C70	2025		pdt_col_1,pdt_col_2,pdt_col_3	; display
			1C70	2026	movl	#1,r0	; return with success
			1C8E	2027	ret		
			1C91	2028	.dsabl	lsb	
			1C92	2029			


```
1C92 2031 .sbttl pdt_byaddr --- display the pdt requested by the user
1C92 2032 :---
1C92 2033 :
1C92 2034 : pdt_byaddr
1C92 2035 :
1C92 2036 : This is a routine whose purpose is display a port
1C92 2037 : descriptor table (PDT) which the user has requested by using
1C92 2038 : the /ADDR qualifier and a valid address of a pdt.
1C92 2039 :
1C92 2040 : Inputs:
1C92 2041 :
1C92 2042 : AP = TPARSE block (TPASL_NUMBER contains the address)
1C92 2043 :
1C92 2044 : Outputs:
1C92 2045 :
1C92 2046 : The requested PDT is displayed if a valid address is specified.
1C92 2047 : Otherwise an informational message is sent to say invalid pdt
1C92 2048 : address.
1C92 2049 : All registers are preserved.
1C92 2050 :
1C92 2051 :---
1C92 2052 :.enabl lsb
1C92 2053 pdt_byaddr::
1C92 2054 .word ^m<r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
1C94 2055 movl tpa$l_number(ap),r7 ; get address of pdt
1C98 2056 jsb verify_pdt ; is this a pdt
1C9E 2057 blbc r0,10$ ; clear, not a pdt
1CA1 2058 :
1CA1 2059 : Now that we are through the validation phase the rest is trivial.
1CA1 2060 :
1CA1 2061 : subhd <VAXcluster data structures> ; set heading
1CAE 2062 skip page
1CB5 2063 pushl r7 ; pass the actual address
1CB7 2064 calls #1, display_pdt ; display it
1CBC 2065 10$:
1CBC 2066 movl #1,r0
1CBF 2067 ret
1CC0 2068
1CC0 2069 verify_pdt:
1CC0 2070 tstl r7 ; check for 80000000 address
1CC2 2071 bgeq 900$ ; not valid
1CC4 2072 addl3 #pdt$b_type,r7,r8 ; point at the type field
1CC8 2073 getmem (r8),r8 ; attempt to read
1CD4 2074 retiferr ; return on error
1CD8 2075 cmpb r8,#dyn$c_scs ; check for the right type
1CDC 2076 bneq 900$ ; not equal, error
1CDE 2077 ashl #-8,r8,r8 ; point at subtype
1CE3 2078 cmpb r8,#dyn$c_scs_pdt ; check for correct subtype
1CE6 2079 bneq 900$ ; not equal, invalid address
1CE8 2080 movl #1,r0 ; valid pdt
1CEB 2081 rsb
1CEC 2082
1CEC 2083 900$: pushl r7 ; invalid pdt address
1CEE 2084 type 1,<!XL is not the address of a PDT>
1D36 2085 movl #0,r0 ; invalid pdt
1D39 2086 rsb
1D3A 2087 .dsabl lsb
```

57 1C AC OFFC
00001CC0'EF 16
1B 50 E9

FEDB CF 57 DD
50 01 FB
58 57 0A C1

60 8F 58 91
OE 12
58 58 F8 8F 78
05 58 91
50 04 12
01 00 D0
05 05


```
1D3A 2089 .sbtcl port descriptor tables & action routines
1D3A 2090 :
1D3A 2091 : PRINT_COLUMNS table for PDT displays
1D3A 2092 :
1D3A 2093 :
1D3A 2094
1D3A 2095 pdt_col_1:
1D3A 2096 column_list
1D3A 2097 pdt$, 20, 8, 2, < -
1D3A 2098 <<Msg Header Size>,l_msghdrsz,ul>,-
1D3A 2099 <<Max Xfer Bcnt>,l_maxbcnt,xl>,-
1D3A 2100 <<DG Header Size>,l_dgovrhd,ul>,-
1D3A 2101 <<Poller Sweep>,l_pollsweep,ul>,-
1D3A 2102 <<Fork Block W.Q.S>,l_waitqfl,q2>,-
1D3A 2103 <<UCB Address>,l_ucb0,xl>,-
1D3A 2104 <<ADP Address>,l_adp,xl>,-
1D3A 2105 <<Accept>,l_accept,xl>,-
1D3A 2106 <<Alloc_Dg_Buf>,l_allocdg,xl>,-
1D3A 2107 <<Alloc_Msg_Buf>,l_allocmsg,xl>,-
1D3A 2108 <<Dealloc_Msg_Buf>,l_deallocmsg,xl>,-
1D3A 2109 <<Dealloc_Msg_Buf_Reg>,l_dealrgmsg,xl>,-
1D3A 2110 >
1EOA 2111 pdt_col_2:
1EOA 2112 column_list
1EOA 2113 pdt$, 15, 8, 2, < -
1EOA 2114 <<Connect>,l_connect,xl>,-
1EOA 2115 <<Dealloc_Dg_Buf>,l_deallocdg,xl>,-
1EOA 2116 <<Disconnect>,l_dconnect,xl>,-
1EOA 2117 <<Unmap>,l_unmap,xl>,-
1EOA 2118 <<Map>,l_map,xl>,-
1EOA 2119 <<Map_Bypass>,l_mapbypass,xl>,-
1EOA 2120 <<Map_Irp>,l_mapirp,xl>,-
1EOA 2121 <<Map_Irp_Bypass>,l_mapirpbyp,xl>,-
1EOA 2122 <<Queue_Dg_Buf>,l_queuedg,xl>,-
1EOA 2123 <<Queue_Mult_Dgs>,l_queuemdgs,xl>,-
1EOA 2124 <<Recycl_Msg_Buf>,l_rclmsgbuf,xl>,-
1EOA 2125 <<Reject>,l_reject,xl>,-
1EOA 2126 >
1EDA 2127 pdt_col_3:
1EDA 2128 column_list
1EDA 2129 pdt$, 17, 8, 0, < -
1EDA 2130 <<Recyclh_Msg_Buf>,l_rchmsgbuf,xl>,-
1EDA 2131 <<Request_Data>,l_reqdata,xl>,-
1EDA 2132 <<Send_Data>,l_senddata,xl>,-
1EDA 2133 <<Send_Dg_Buf>,l_senddg,xl>,-
1EDA 2134 <<Send_Msg_Buf>,l_sendmsg,xl>,-
1EDA 2135 <<Send_Cnt_Msg_Buf>,l_sndcntmsg,xl>,-
1EDA 2136 <<Read_Count>,l_readcount,xl>,-
1EDA 2137 <<Rls_Read_Count>,l_rlscount,xl>,-
1EDA 2138 <<Mreset>,l_mreset,xl>,-
1EDA 2139 <<Mstart>,l_mstart,xl>,-
1EDA 2140 <<Stop_Vcs>,l_stop_vcs,xl>,-
1EDA 2141 <<Send_Dg_Reg>,l_sendrgdg,xl>,-
1EDA 2142 >
1FAA 2143 .end
```


CLUSTER
Symbol table

SHOW CLUSTER INFORMATION

E 3

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 54
(31)

\$\$\$	= 000002F5	R	04	CDTSW_BLKSTATE	= 0000002A		
\$\$TMP1	= 00000001			CDTSW_INITLREC	= 00000048		
\$\$TMP2	= 000000EF			CDTSW_MINREC	= 00000044		
\$\$T2	= 00000005			CDTSW_PENDREC	= 00000046		
ADD_SYMBOL	*****	X	03	CDTSW_QBDT_CNT	= 0000009A		
ARG\$	= 00000001			CDTSW_QCR_CNT	= 00000098		
CDL	= 00000000	R	02	CDTSW_REASON	= 00000026		
CDL\$L_FREECDT	= FFFFFFFF4			CDTSW_REC	= 00000042		
CDL\$W_MAXCONIDX	= FFFFFFFF0			CDTSW_SEND	= 00000040		
CDL\$W_SIZE	= FFFFFFFF8			CDTSW_STATE	= 00000028		
CDL_SIZE	= 00000004	R	02	CDT_6BYTES	000017D7	R	03
CDR\$SL_CDT	= 00000024	R		CDT_BLKSTATE	000002C8	R	03
CDT	= 00000008	R	02	CDT_BYADDR	0000152E	RG	03
CDT\$B_RSTATION	= 00000020			CDT_COL_1	000015C9	R	03
CDT\$B_TYPE	= 0000000A			CDT_COL_2	00001689	R	03
CDT\$C_ACCP_PEND	= 00000002			CDT_COL_3	00001739	R	03
CDT\$C_ACCP_SENT	= 0000000A			CDT_FAQ_6BYTES	000017C9	R	03
CDT\$C_CLOSED	= 00000000			CDT_SPCFY	000006F9	RG	02
CDT\$C_CON_ACK	= 00000008			CDT_STATE	00000258	R	03
CDT\$C_CON_PEND	= 00000001			CLUGL_CLUB	*****	X	03
CDT\$C_CON_REC	= 00000009			CLUB	000000A8	R	02
CDT\$C_CON_SENT	= 00000007			CLUB\$B_CLUFCB	= 0000010C		
CDT\$C_CR_PEND	= 00000005			CLUB\$B_CUR_CODE	= 00000058		
CDT\$C_DCR_PEND	= 00000006			CLUB\$B_CUR_PHASE	= 00000059		
CDT\$C_DISC_ACK	= 00000003			CLUB\$B_FSYSID	= 00000026		
CDT\$C_DISC_MTCB	= 00000006			CLUB\$B_LST_CODE	= 00000044		
CDT\$C_DISC_PEND	= 00000004			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_DISC_REC	= 00000004			CLUB\$C_LST_CODE	= 00000044		
CDT\$C_DISC_SENT	= 00000005			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_LENGTH	= 000000A0			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_LISTEN	= 00000001			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_OPEN	= 00000002			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_REJ_PEND	= 00000003			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_REJ_SENT	= 0000000B			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_VC_FAIL	= 0000000C			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_AUXSTRUC	= 0000005C			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_BYTMAPD	= 00000094			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_BYTREQD	= 00000090			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_BYTSENT	= 00000088			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_CONDAT	= 00000058			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_CRWAITQFL	= 00000038			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_DGDISCARD	= 00000078			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_DGRCVD	= 00000074			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_DGSENT	= 00000070			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_ERRADDR	= 0000000C			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_LCONID	= 00000018			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_LPROCNAM	= 00000054			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_MSGRCVD	= 00000080			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_MSGSENT	= 0000007C			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_PB	= 0000001C			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_PDT	= 00000010			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_RCONID	= 00000014			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_REQDATS	= 0000008C			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_RPROCNAM	= 00000050			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_SCSMSG	= 0000002C			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_SNDATS	= 00000084			CLUB\$C_LENGTH	= 000001A8		
CDT\$C_WAITQFL	= 00000030			CLUB\$C_LENGTH	= 000001A8		

CLUSTER
Symbol table

SHOW CLUSTER INFORMATION

F 3

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 55
(31)

```

CLUBSV_SHUTDOWN      = 00000002
CLUBSV_STS_PH0       = 00000009
CLUBSV_STS_PH1       = 0000000B
CLUBSV_STS_PH1B      = 0000000A
CLUBSV_STS_PH2       = 0000000C
CLUBSV_STS_PPHASE    = 00000008
CLUBSV_TRANSITION    = 0000001D
CLUBSV_UNLOCK        = 00000011
CLUBSW_MEMSEQ        = 000000AC
CLUBSW_MSGCNT        = 0000005A
CLUBSW_NEXT_CSID     = 00000064
CLUBSW_NODES         = 00000024
CLUBSW_QDVOTES       = 000000AE
CLUBSW_QUORUM        = 00000020
CLUBSW_VOTES         = 00000022
CLUB_2WORDS          = 00000653 R 03
CLUB_6BYTES          = 00000860 R 03
CLUB_COL_1           = 00000664 R 03
CLUB_COL_2           = 00000754 R 03
CLUB_FAO_6BYTES      = 00000641 R 03
CLUB_FLAGS           = 00000148 R 03
CLUB_SUMMARY         = 00000000 R 03
CLUCB                = 000002FC R 02
CLUCBSB_COUNTER      = 00000024
CLUCBSB_LENGTH       = 00000229
CLUCBSL_ACT_COUNT    = 00000018
CLUCBSL_IRP          = 00000010
CLUCBSL_QFLBN        = 0000001C
CLUCBSL_TQE          = 00000014
CLUCBSL_UCB          = 0000000C
CLUCBSV_QF_CSPACK    = 00000004
CLUCBSV_QF_ERROR     = 00000003
CLUCBSV_QF_RIP       = 00000001
CLUCBSV_QF_TIM       = 00000000
CLUCBSV_QF_WIP       = 00000002
CLUCBSV_QS_ACTIVE    = 00000002
CLUCBSV_QS_CLUSTER   = 00000003
CLUCBSV_QS_NOT_READY = 00000000
CLUCBSV_QS_READY     = 00000001
CLUCBSV_QS_VOTE      = 00000004
CLUCBSW_FLAGS        = 00000022
CLUCBSW_STATE        = 00000020
CLUCB_FLAGS          = 00000228 R 03
CLUCB_STATE          = 000001F8 R 03
CLUCBSL_ID           = 0000001C
CLUCBSL_STATUS       = 00000020
CLUCBSL_STEP         = 00000018
CLUCBSL_SYNC_CSB     = 00000024
CLUCBSV_ACTIVE       = 00000000
CLUCBSV_FKB_BUSY     = 00000003
CLUCBSV_PENDING      = 00000001
CLUCBSV_SYNC_NODE    = 00000002
CLUSTER_SUMMARY      = 0000043F R 03
CMND_BUFFER          = ***** X 03
CMND_DESCR           = ***** X 03
COLMSK_FAO_AC        = 00000000
COLMSK_FAO_AS        = 00000001

```

```

COLMSK_FAO_Q2        = 00000011
COLMSK_FAO_UB        = 00000005
COLMSK_FAO_UL        = 0000000F
COLMSK_FAO_UW        = 0000000A
COLMSK_FAO_XB        = 00000003
COLMSK_FAO_XL        = 0000000D
COLMSK_FAO_XW        = 00000008
COLMSK_LENGTH        = 00000010
COUNT_PATHS         = 0000106B R 03
CSB                  = 00000250 R 02
CSBSB_ECOLVL         = 00000040
CSBSB_REF_CNT        = 0000006C
CSBSB_REMACKLIM      = 00000033
CSBSB_STATE          = 00000043
CSBSB_UNACKEDMSGS    = 00000032
CSBSB_VERNUM         = 00000041
CSBSC_LENGTH         = 000000AC
CSBSK_ACCEPT         = 00000006
CSBSK_CONNECT        = 00000005
CSBSK_DEAD           = 0000000A
CSBSK_DISCONNECT     = 00000007
CSBSK_LOCAL          = 0000000B
CSBSK_NEW            = 00000004
CSBSK_OPEN           = 00000001
CSBSK_REACCEPT       = 00000008
CSBSK_RECONNECT      = 00000003
CSBSK_STATUS         = 00000002
CSBSK_WAIT           = 00000009
CSBSL_CDT            = 0000000C
CSBSL_CSID           = 0000004C
CSBSL_CURRCDRP       = 00000034
CSBSL_PARTNERQFL     = 00000058
CSBSL_PDT            = 00000010
CSBSL_RESENDQFL      = 0000001C
CSBSL_SB             = 00000068
CSBSL_SENTQFL        = 00000014
CSBSL_STATUS         = 00000060
CSBSL_SYSQFL         = 00000000
CSBSL_TIMEOUT        = 00000048
CSBSL_TQE            = 00000044
CSBSQ_REFTIME        = 00000074
CSBSQ_SWINCARN       = 00000038
CSBSV_CLUSTER        = 00000008
CSBSV_LOCAL          = 00000018
CSBSV_LOCKED         = 00000010
CSBSV_LONG_BREAK     = 00000000
CSBSV_MEMBER         = 00000001
CSBSV_QF_ACTIVE      = 00000009
CSBSV_QF_SAME        = 00000003
CSBSV_REMOVED        = 00000002
CSBSV_SELECTED       = 00000011
CSBSV_SEND_STATUS    = 0000001A
CSBSV_SHUTDOWN       = 0000000A
CSBSV_STATUS_RCVD    = 00000019
CSBSW_ACKRSEQNM      = 00000030
CSBSW_LCKDIRWT       = 00000054
CSBSW_QDVOTES        = 00000056

```


CLUSTER
Symbol table

SHOW CLUSTER INFORMATION

G 3

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1Page 56
(31)

CSBSW_QUORUM	= 00000052		NOTRANS	00000C67	R	03
CSBSW_RCVDSQNM	= 0000002E		NO_RSPID	0000198D	R	03
CSBSW_SENDSEQNM	= 0000002C		NUCL_STRING	000010C9	R	03
CSBSW_VOTES	= 00000050		OUTPUT	*****	X	03
CSBQUOR_VOTES	00000E9F	R	PAGE_SIZE	*****	X	03
CSB_2BYTES	00000CDF	R	PBSL_FLINK	= 00000000		
CSB_2WORDS	00000CCF	R	PBSL_SBLINK	= 00000030		
CSB_COL_1	00000CEF	R	PDT	00000525	R	02
CSB_COL_2	00000D7F	R	PDT\$B-PDT TYPE	= 00000007		
CSB_COL_3	00000E0F	R	PDT\$B-TYPE	= 0000000A		
CSB_STATES	00000058	R	PDT\$C-LENGTH	= 000000E4		
CSB_STATUS	000000B8	R	PDT\$C-PA	= 00000001		
CSB_SUMMARY	00000028	R	PDT\$C-PE	= 00000003		
CSID	000006F5	RG	PDT\$C-PS	= 00000004		
CURR_DATE	000008F5	R	PDT\$C-PU	= 00000002		
CURR_TIME	000008FE	R	PDT\$C-PU	= 00000002		
DATE_ROUTINE	0000087E	R	PDT\$C-PU	= 0000000C		
DCB_COL_1	00000AF2	R	PDT\$C-PU	= 0000000C		
DCB_COL_2	00000B32	R	PDT\$C-PU	= 000000E0		
DDB\$B_DRVNAME_LEN	= 00000024		PDT\$C-PU	= 00000010		
DDB\$B_NAME_LEN	= 00000014		PDT\$C-PU	= 00000014		
DDB\$T_DRVNAME	= 00000024		PDT\$C-PU	= 00000018		
DDB\$T_NAME	= 00000014		PDT\$C-PU	= 00000028		
DEVICE_NAME	000006D9	R	PDT\$C-PU	= 0000001C		
DIRECTORY	00000609	R	PDT\$C-PU	= 00000020		
DISPLAY_CDT	000013A9	RG	PDT\$C-PU	= 00000024		
DISPLAY_CLUB	000005A8	R	PDT\$C-PU	= 000000B8		
DISPLAY_CLUDCB	000009D6	R	PDT\$C-PU	= 00000000		
DISPLAY_CLUFCB	0000093D	R	PDT\$C-PU	= 0000002C		
DISPLAY-CSB	00000B72	R	PDT\$C-PU	= 00000030		
DISPLAY-PDT	00001B97	RG	PDT\$C-PU	= 00000034		
DISPLAY-RD_ENTRY	00001995	RG	PDT\$C-PU	= 00000038		
DISPLAY-SB_PBS	00001091	R	PDT\$C-PU	= 000000BC		
DISPLAY-SUMLINE	00001281	R	PDT\$C-PU	= 00000070		
DONE	000005A4	R	PDT\$C-PU	= 000000B4		
DRIVER_NAME	000006C5	R	PDT\$C-PU	= 00000074		
DYN\$C_SCS	= 00000060		PDT\$C-PU	= 000000D8		
DYN\$C_SCS_CDT	= 00000002		PDT\$C-PU	= 0000003C		
DYN\$C_SCS-PDT	= 00000005		PDT\$C-PU	= 00000040		
ECO_VERS	00000EBB	R	PDT\$C-PU	= 00000044		
FAB\$L_STV	*****	X	PDT\$C-PU	= 00000048		
FCB_COL_1	00000AA2	R	PDT\$C-PU	= 00000068		
FCB_COL_2	00000AD2	R	PDT\$C-PU	= 0000004C		
FCB_STATUS	00000120	R	PDT\$C-PU	= 00000050		
FIND_PROCNAME	000012EB	RG	PDT\$C-PU	= 0000006C		
FREE_CDT_LIST	00001258	R	PDT\$C-PU	= 00000054		
GETMEM	*****	X	PDT\$C-PU	= 00000058		
LIB\$FREE_VM	*****	X	PDT\$C-PU	= 0000005C		
LIB\$GET_VM	*****	X	PDT\$C-PU	= 0000007C		
LIB\$SIGNAL	*****	X	PDT\$C-PU	= 00000060		
LINE_COUNT	*****	X	PDT\$C-PU	= 00000080		
LOCATE-CSB	000003E5	R	PDT\$C-PU	= 000000DC		
LOOP	0000050E	R	PDT\$C-PU	= 00000064		
MAKE-CSB_SYMBOLS	00000BEB	R	PDT\$C-PU	= 000000AC		
MSG\$SUCCESS	*****	X	PDT\$C-PU	= 00000000		
NEW_PAGE	*****	X	PDT\$C-PU	= 00000004		
NODE	000006A5	R	PDT\$C-PU	= 00001C92	RG	03
			PDT\$C-PU	00001D3A	R	03
			PDT\$C-PU	00001E0A	R	03

CLUSTER
Symbol table

SHOW CLUSTER INFORMATION

H 3

16-SEP-1984 01:24:07 VAX/VMS Macro V04-00
5-SEP-1984 03:31:48 [SDA.SRC]CLUSTER.MAR;1

Page 57
(31)

PDT_COL_3	00001EDA	R	03
PDT_TYPE	00000300	R	03
PORT_CHAR	00000328	R	03
PRINT	*****	X	03
PRINT_COLUMNS	*****	X	03
PRINT_COLUMN_VALUE	*****	X	03
PROCNAME	000006B5	R	02
QUOR_VOTE	00000844	R	03
RAB\$C_RBF	*****	X	03
RAB\$W_RSZ	*****	X	03
RDSV_BUSY	= 00000000		
RDT	00000639	R	02
RDT\$C_LENGTH	= 00000018		
RDT\$C_FREERD	= FFFFFFFF4		
RDT\$C_MAXRDIDX	= FFFFFFFF8		
RDT\$C_WAITFL	= FFFFFFFE8		
RDT\$W_SIZE	= FFFFFFFF0		
RDT_SIZE	0000063D	R	02
REMOTE_NODE	0000136D	R	03
SB\$B_SYSTEMID	= 00000018		
SB\$C_LENGTH	= 00000060		
SB\$C_FLINK	= 00000000		
SB\$C_PBFL	= 0000000C		
SB\$S_NODENAME	= 00000010		
SB\$T_NODENAME	= 00000044		
SB\$T_SWTYPE	= 00000024		
SBLOCK	00000645	R	02
SCS\$GL_CDL	*****	X	03
SCS\$GL_PDT	*****	X	03
SCS\$GL_RDT	*****	X	03
SCS\$GQ_CONFIG	*****	X	03
SCS\$GQ_DIRECT	*****	X	03
SCS_SUMMARY	00000F06	R	03
SDIR\$B_PROCLNF	= 0000001C		
SDIR\$B_PROCNAM	= 0000000C		
SDIR\$C_LENGTH	= 00000030		
SDIR\$C_CONID	= 0000002C		
SDIR\$C_FLINK	= 00000000		
SET_HEADING	*****	X	03
SHOW_CLUSTER	00000338	RG	03
SHOW_CONNECTIONS	000010CA	RG	03
SHOW_PORTS	00001A25	RG	03
SHOW_RSPID	000017F4	RG	03
SHOW_SCS	00000ED7	RG	03
SHOW_SYSTEM_BLOCK	*****	X	03
SKIP_LINES	*****	X	03
STATE_TRANSLATE	000012CA	R	03
SYSS\$ACTIM	*****	GX	03
SYSS\$FAO	*****	X	03
SYSS\$PUT	*****	GX	03
TIME_ROUTINE	000008B6	R	03
TIM_BUFFER	000006ED	R	02
TPASL_NUMBER	= 0000001C		
TRANSLATE_ADDRESS	*****	X	03
TRANSLATE_BITS	*****	X	03
TRANS_BYTE	0000092B	R	03
TRANS_LONG	00000907	R	03

TRANS_WORD
UCB\$C_DDB
UCB\$W_UNIT
VERIFY_CDT
VERIFY_PDT
WAIT_CDRP

00000919	R	03
= 00000028		
= 00000054		
0000154F	R	03
00001CC0	R	03
00000641	R	02

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABSS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
\$DATA	000006FD (1789.)	02 (2.)	NOPIC USR CON REL LCL NOSHR NOEXE RD WRT NOVEC BYTE
CLUSTER	00001FAA (8106.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC LONG
LITERALS	0000149C (5276.)	04 (4.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC BYTE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	35	00:00:00.06	00:00:00.77
Command processing	136	00:00:00.42	00:00:04.62
Pass 1	649	00:00:20.19	00:01:09.47
Symbol table sort	0	00:00:01.73	00:00:05.20
Pass 2	418	00:00:05.87	00:00:21.36
Symbol table output	41	00:00:00.24	00:00:00.71
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1283	00:00:28.53	00:01:42.16

The working set limit was 2100 pages.
189601 bytes (371 pages) of virtual memory were used to buffer the intermediate code.
There were 90 pages of symbol table space allocated to hold 1388 non-local and 314 local symbols.
2143 source lines were read in Pass 1, producing 70 object records in Pass 2.
50 pages of virtual memory were used to define 46 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
-\$255\$DUA28:[SDA.OBJ]SDALIB.MLB;1	17
-\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	15
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	10
TOTALS (all libraries)	42

1632 GETS were required to define 42 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:CLUSTER/OBJ=OBJ\$:CLUSTER MSRC\$:CLUSTER/UPDATE=(ENH\$:CLUSTER)+EXECMLS/LIB+LIB\$:SDALIB/LIB

0350

AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

0351 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400
401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600
601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700
701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800
801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900
901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000